CodeWarrior®

Targeting the Java VM From Windows

Because of last-minute changes to CodeWarrior, some of the information in this manual may be inaccurate. Please read the Release Notes on the CodeWarrior CD for the latest up-to-date information.

Revised: 98/08/28 dcb

Metrowerks CodeWarrior copyright ©1993–1998 by Metrowerks Inc. and its licensors. All rights reserved.

Documentation stored on the compact disk(s) may be printed by licensee for personal use. Except for the foregoing, no part of this documentation may be reproduced or transmitted in any form by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system, without permission in writing from Metrowerks Inc.

Metrowerks, the Metrowerks logo, CodeWarrior, and Software at Work are registered trademarks of Metrowerks Inc. PowerPlant and PowerPlant Constructor are trademarks of Metrowerks Inc.

All other trademarks and registered trademarks are the property of their respective owners.

ALL SOFTWARE AND DOCUMENTATION ON THE COMPACT DISK(S) ARE SUBJECT TO THE LICENSE AGREEMENT IN THE CD BOOKLET.

How to Contact Metrowerks:

U.S.A. and international	Metrowerks Corporation 9801 Metric, Suite 100 Austin, TX 78758 U.S.A.
Canada	Metrowerks Inc. 1500 du College, Suite 300 Ville St-Laurent, QC Canada H4L 5G6
Ordering	Voice: (800) 377–5416 Fax: (512) 873–4901
World Wide Web	`http://www.metrowerks.com`
Registration information	`register@metrowerks.com`
Technical support	`support@metrowerks.com`
Sales, marketing, & licensing	`sales@metrowerks.com`
CompuServe	go `Metrowerks`

Table of Contents

1 Introduction **7**

Read the Release Notes 7
Metrowerks Year 2000 Compliance 7
What's New in This Release 8
What's in This Book 9
Where to Go from Here 9

2 Getting Started with Codewarrior for Java **11**

System Requirements 11
Installing CodeWarrior for Java 11
Overview of Java in CodeWarrior 12
Development Tools for Java 15
CodeWarrior IDE 15
Metrowerks Java Compiler 16
Metrowerks JIT Compiler 16
Java Linker 17
CodeWarrior Debugger 17
Java API Headers 17

3 Programming Tutorial for Java **19**

Applet Description 19
Starting Out 20
Creating a project 21
Changing Target Settings 24
Writing the Applet 29
Adding the Java File 29
Add Graphics Files 32
Editing the HTML File 38
Compile and Run 41
Oops! What's Happening? 45
Using the CodeWarrior Debugger 46
Exercises 52

4 Creating Java Projects **55**
Types of Java Projects . . . 55
Applets . . . 55
Applications . . . 56
Libraries . . . 56
Using Project Stationery . . . 56
Working with Java in CodeWarrior . . . 57
Creating a New Java Project . . . 58
Creating Java Code . . . 60
Changing Settings . . . 61
Running a Java Project . . . 62
Debugging a Java Project . . . 63
Kinds of Application Projects . . . 64
Using the classes.zip Library . . . 64

5 JavaDoc **67**
CodeWarrior JavaDoc Implementation . . . 67
Using JavaDoc . . . 67

6 Target Settings for Java **73**
Target Settings . . . 74
Java Target . . . 76
Applet . . . 77
Application . . . 77
Library . . . 79
Java Command Line . . . 79
Java Language . . . 80
FTP Post Linker . . . 84
Java Output . . . 86
Class Folder . . . 87
Jar File . . . 88
JavaDoc . . . 89

7 Debugging Java Projects **93**
Debugger Features and Limitations . . . 93
Special Debugger Features for Java . . . 94

Breaking on Java Exceptions 94
Opening Multiple Class Files in One Browser 95
Choosing a Java Applet Viewer for Debugging 95
Debugging Threads. 95
Viewing the Java Virtual Machine Disassembly. 96
Specifying Java Debugger Settings 97

8 Troubleshooting on Java **101**
Programming Problems 101
Conversion Problems 101

Index **103**

1 Introduction

This chapter is your introduction to targeting the Java Virtual Machine (VM) with CodeWarrior. It includes the following topics:

- Read the Release Notes
- Metrowerks Year 2000 Compliance
- What's New in This Release
- What's in This Book
- Where to Go from Here

Read the Release Notes

Before using CodeWarrior Java, read the release notes. They contain important information about any late-breaking changes.

If you are new to Java in CodeWarrior, or to CodeWarrior in general, we strongly recommend that you read this chapter carefully. You'll get a high-level overview of Java in the CodeWarrior environment. This includes a short, hands-on tutorial.

Metrowerks Year 2000 Compliance

The Products provided by Metrowerks under the License agreement process dates only to the extent that the Products use date data provided by the host or target operating system for date representations used in internal processes, such as file modifications. Any Year 2000 Compliance issues resulting from the operation of the Products are therefore necessarily subject to the Year 2000 Compliance of the relevant host or target operating system. Metrowerks directs you to the relevant statements of Microsoft Corporation, Sun Microsystems, Inc., Apple Computer, Inc., and other host or target operating systems relating to the Year 2000 Compliance of their operating sys-

tems. Except as expressly described above, the Products, in themselves, do not process date data and therefore do not implicate Year 2000 Compliance issues.

For additional information, visit: `http://www.metrowerks.com/about/y2k.html`.

What's New in This Release

There are a lot of changes in this latest version of Java for CodeWarrior, to help you program more efficiently:

- The JavaDoc tool is now packaged in a pre linker. This increases performance since the pre linker does not have to parse source files while generating docs.
- You can now debug Java applets and applications running on any of the following virtual machines: Apple MRJ, MW JIT, Sun JDK, Microsoft VM.
- The Java Command Line post linker has been added to allow you to run command line post linker tools when building your Java projects.
- Changing project settings, adding and removing files, and similar operations are now faster when using the Java Linker.
- JavaDoc can now optionally output its docs as a folder heirarchy rather than a flat folder hierarchy with long filenames. See "Output as folder hierarchy" on page 90 for details.
- JavaDoc can now optionally generate Macintosh-friendly filenames shortened to 31 characters to allow for cross-platform html docs. See "Generate Mac-friendly filenames" on page 89 for more information.
- The JavaDoc plugin now uses the Sun JDK or JRE 1.1.6. Therefore, Sun JDK or JRE 1.1.6 must be installed in the system folder in order for this plugin to work.

Many other improvements have been made to the IDE. See the *IDE User Guide* for more information.

What's in This Book

General description of the book's contents by, summarized in a table (Table 1.1) with cross-references by introductory topic.

Table 1.1 Contents of chapters

Introductory Topic	Description
Introduction	Contents of this manual; general description of the CodeWarrior development tools; where to go next
Getting Started with Codewarrior for Java	Installation and setup for Java
Programming Tutorial for Java	Tutorial on Java programming
Creating Java Projects	Creating Java programs
JavaDoc	How to set up and use JavaDoc
Target Settings for Java	Project settings specific to Java programming
Debugging Java Projects	Debugging Java code with the CodeWarrior Debugger
Troubleshooting on Java	Troubleshooting information specific to Java

Where to Go from Here

The information in this manual is, for the most part, specific to either the Java language or the Java virtual machine as a target. This manual does not cover basic features of either the CodeWarrior IDE or the CodeWarrior Debugger. This manual contains Java-specific information.

You do not have to read the chapters in this manual sequentially. Use this manual as a reference to learn about Java in CodeWarrior, or to answer questions you encounter as you develop Java code.

You'll find all the manuals mentioned in this section on the CodeWarrior Reference CD.

For everyone:

- See the *IDE User Guide* and the *Debugger User Guide* for complete information about the CodeWarrior Integrated Development Environment and debugger.

If you're new to CodeWarrior:

- Look for the CodeWarrior tutorials. You'll find them on your CodeWarrior Reference CD.

For general information on Java programming:

This manual does not teach Java syntax, nor does it introduce you to the classes and methods in the Java API—the calls you use to program the Java virtual machine.

- *Discover Programming* from Metrowerks can teach you how to program in Java. You can learn more about Discover Programming at:

  `http://www.metrowerks.com/discover/`

- To learn Java, you may consult the *Java Language Tutorial*. This HTML document is available directly from Sun Microsystems at:

  `http://java.sun.com/doc.html`

- To learn more about the Java API, consult the *Java API Documentation*, available as part of the CodeWarrior documentation. This document is in HTML format and comes from Sun Microsystems.

There are also many third party books on Java programming. Check your local bookstore or favorite online bookstore for a wide selection.

2

Getting Started with Codewarrior for Java

This chapter gives you a brief overview of Java in CodeWarrior, installation, system requirements and tool available to you.

This chapter includes the following topics:

- System Requirements
- Installing CodeWarrior for Java
- Overview of Java in CodeWarrior
- Development Tools for Java

System Requirements

If you can run CodeWarrior, you can target the Java virtual machine. The Windows-hosted version of CodeWarrior requires a 80486DX or Pentium™-class processor, 16 megabytes of RAM, Microsoft Windows 95 or Windows NT 4.0 operating system, and a CD-ROM drive to install the software.

For optimum performance, we recommend that you use a computer equipped with a Pentium™-class processor with at least 24 megabytes of RAM.

Installing CodeWarrior for Java

Use the CodeWarrior installer application and follow the instructions in that application. The installer ensures that everything is installed in the proper locations.

Overview of Java in CodeWarrior

Java is both a programming language and, in a virtual sense, a computer platform—a target, in CodeWarrior terminology. As such, Java is unique among the languages and targets supported in the CodeWarrior IDE. You can think of Java as not only a language, but as an instruction set for the Java *virtual machine*.

The virtual machine, commonly called a VM, is an abstract computer microprocessor. Like any silicon computer chip, the virtual machine has an instruction set. The instructions for the Java virtual machine are called *bytecodes*.

A bytecode is simply a stream of formatted bytes that has a precisely defined impact on the virtual machine. For example, there are bytecodes that are branch instructions, that push and pop values on a stack, that load or store values, and so forth. Just like traditional assembly language instructions affect a silicon-based processor, the Java bytecodes are a real instruction set for the Java virtual machine.

The reason for the abstract nature of the virtual machine and its bytecode instruction set is to allow you, the programmer, to create platform-independent code. It is the responsibility of the runtime interpreter (or just-in-time compilers that convert bytecodes to native code on the fly) to replicate in a real operating system the effect that a bytecode has on the virtual machine.

If you are an experienced code warrior, you'll see that what you do in the Java environment is a little different from what you have done in the past using C, C++, or Pascal.

The development process for Java software in CodeWarrior has the following basic steps.

1. **Write source code.**

   In CodeWarrior, you write Java source code in exactly the same way you do for C, C++, and Pascal. You use the same project manager and the same source code editor. You get all the power and features of CodeWarrior. To target the Java virtual machine, you simply make the appropriate selection in the proper settings panel.

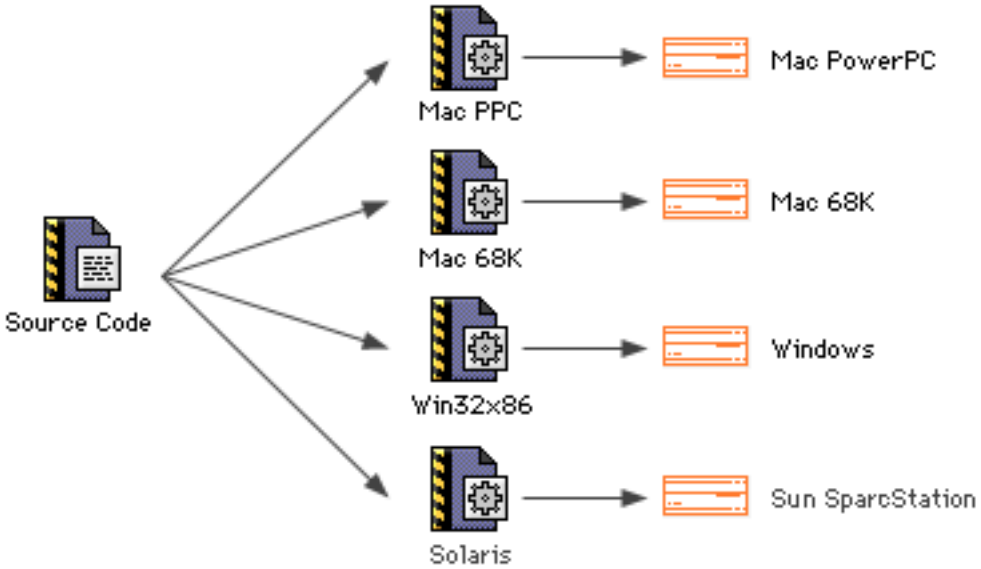
2. **Compile the source code.**

   Again, this process is exactly like it is for any language and target supported in CodeWarrior. You simply issue the appropriate **Compile** or **Make** command from the CodeWarrior Project menu. In response, the CodeWarrior Java compiler generates Java bytecodes based on your source code. CodeWarrior saves the bytecodes in a Java class file.

   The bytecodes in the Java class file are analogous to the object code generated by a C/C++ or Pascal compiler. The difference between traditional object code and Java bytecodes is in how they run.

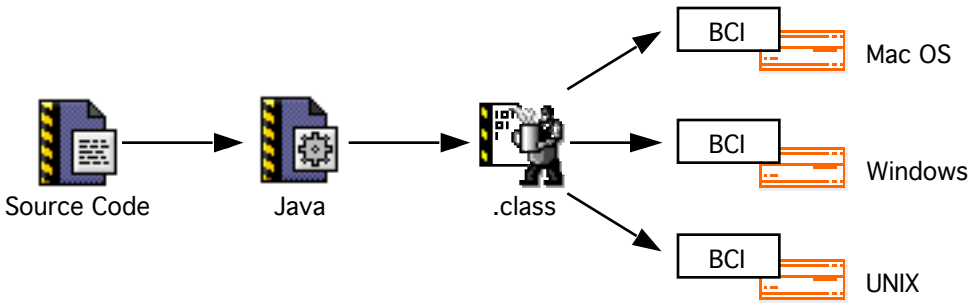
   Object code can run directly on the platform for which it was compiled, but cannot run on any other platform. You must compile it over and over again for each platform. There are also serious cross-platform problems you can encounter trying to address multiple operating systems and computer microprocessors.

Figure 2.1 Compiling and running object code



Java bytecodes cannot run directly on any real machine. On the other hand, the Java bytecodes created by the CodeWarrior Java compiler can be interpreted to run on *any* platform that has a bytecode interpreter (BCI). You write the code once, compile it once, and then you're done.

Figure 2.2 Compiling and running Java bytecodes



3. **Run the code.**

   Just choose **Run** from the Project menu and CodeWarrior runs the project using the specified applet viewer (an HTML browser) or the jview utility (for a Java application). The default applet viewer is Internet Explorer. To learn how to specify a different viewer, see "Target Settings" on page 74.

   Remember, the bytecodes in the class file cannot run directly on any real platform. The class file must be run under the control of a bytecode interpreter (BCI). The BCI is responsible for translating the bytecodes into machine-native instructions.

4. **Debug the code with the CodeWarrior Debugger.**

   The final step in the development process is to test and debug your code. You debug Java code using the same CodeWarrior debugger you use for C/C++ and Pascal code. The CodeWarrior debugger understands Java. The debugger does everything you would expect the debugger to do. It has a stack crawl, displays local variables, and allows you to set breakpoints.

   To debug Java code, your applet or application must be built to include debugging information. Choose **Enable Debugger** from the Project menu to tell the Java compiler that you want to create debugging information when it compiles your code. Then recompile your project. You can choose **Make** from the Project menu to compile and link your code. You may also choose **Debug** from the Project menu, and CodeWarrior compiles, links, and launches your program under debugger control.

   From here you can set break points, and run your program.

Summary

That's all there is to it. As you can see, the big difference between Java and compiled languages is that Java code is interpreted at runtime.

If you have used CodeWarrior in the past, developing Java code in CodeWarrior is going to look very familiar to you. If you are new to CodeWarrior, you'll find developing Java code to be a pleasure once you master the tools.

Development Tools for Java

These are the tools CodeWarrior provides for developing Java software:

- CodeWarrior IDE
- Metrowerks Java Compiler
- Metrowerks JIT Compiler
- Java Linker
- CodeWarrior Debugger
- Java API Headers

CodeWarrior IDE

The CodeWarrior Integrated Development Environment (IDE) provides a complete set of tools for developing application programs for a variety of target platforms, including 68K- and PowerPC-based Mac OS systems, Win32/x86 systems, Solaris, and Java. You use the same IDE when developing code for all target platforms; you designate the platform and code generator of your choice when you create your project.

For information on the CodeWarrior IDE, see the *IDE User Guide*.

Metrowerks Java Compiler

The Metrowerks Java compiler is a plug-in tool built exclusively for use in the CodeWarrior IDE.

Metrowerks JIT Compiler

The Metrowerks Just-In-Time compiler (JIT), included as part of the Java Compiler, seeks to accelerate Java execution by translating Java bytecodes into native machine code. It does this on an incremental, method by method basis. The first time a method is invoked, the JIT compiler is run instead, transforming it into native code. Once transformed, this native code runs in place of the bytecodes. For all future invocations during the lifetime of the Java VM session, the native code is run instead.

The JIT compiler does some simple optimizations in order to improve the quality of the code it produces. One of the most significant features is, mapping the Java stack to registers. With this feature, it is often possible to map several Java VM instructions to one native machine instruction. The JIT compiler knows about a number of frequently occurring patterns which it rewrites into sequences to produce better code.

NOTE: There are some commonly used Java VM instructions that are large, resulting in a 12:1 code expansion over bytecodes.

The JIT compiler also performs Common Subexpression Elimination (CSE) for arrays. This means the JIT does not reevaluate duplicate array indices.

For example, if you have two or more occurrences of `a[i]` where `i=2`, the JIT evaluates the first occurrence of `a[2]` then uses that value in place of all other occurrences of `a[2]`.

Java Linker

Like the compilers, the Java Linker is a plug-in tool integrated into the CodeWarrior IDE.

CodeWarrior Debugger

The CodeWarrior debugger controls your program's execution and allows you to see what's happening internally as your program runs. You use the debugger to find problems in your program's execution.

The debugger can execute your program one statement at a time, and suspend execution when control reaches a specified point. When the debugger stops a program, you can view the chain of function calls, examine and change the values of variables, and inspect the contents of the processor's registers.

For general information about the debugger, including all of its common features and its visual interface, you should read the *Debugger User Guide*.

For more information about debugging software for Java, see "Debugging Java Projects" on page 93.

Java API Headers

The Java Application Programming Interface (API) is a set of data structures and functions used to interface to the Java operating system.

3

Programming Tutorial for Java

This tutorial takes you very quickly through the CodeWarrior Java environment. This tutorial does not teach you Java programming. It is designed to teach you how to use the CodeWarrior Integrated Development Environment (IDE) to write and debug Java code. This tutorial takes you step-by-step through the entire process. The applet you create is designed to be immediately reusable with little to no change, depending on your needs.

NOTE: This tutorial requires a Java enabled browser or Java applet viewer. If you do not already have a Java enabled browser installed, the latest version of Microsoft Internet Explorer is available on the CodeWarrior CD. We recommend you install it before continuing with this tutorial.

The topics discussed in this tutorial are:

- Applet Description
- Starting Out
- Writing the Applet
- Compile and Run
- Oops! What's Happening?
- Exercises

Applet Description

The applet you will create in this tutorial is a simple navigation menu, called an Image Map, that you can add to any web page. It consists of two graphics, one for the menu in its unhighlighted state,

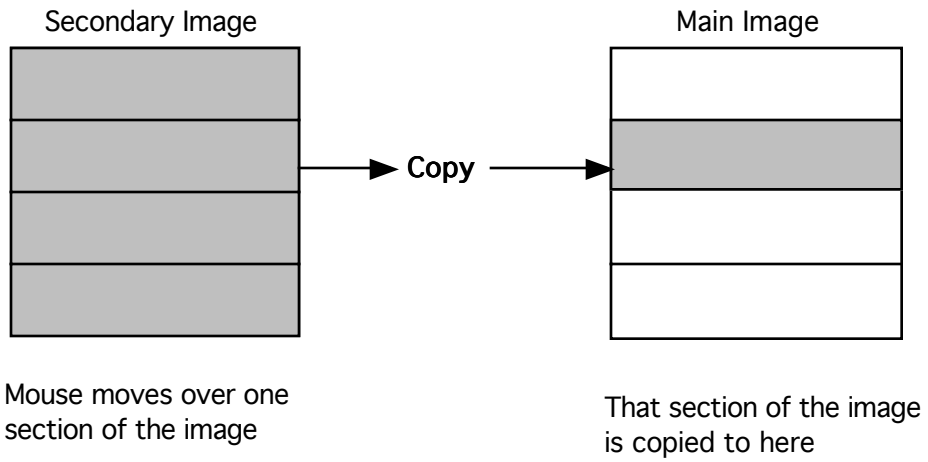
and one which will be used to draw each menu item in its highlighted state. The original idea for this tutorial comes from a similar applet written by Chris Cason on the Pov-Ray home page located at:

`http://www.povray.org`.

Theory of Operation

How the applet works is very simple. It loads two graphics into memory. The first graphic is an image of the menu, listing all choices. The second graphic is the same image with some minor differences (in this case color) to show the current menu choice. As the mouse passes over each option in the menu, the applet copies the corresponding area from the secondary graphic and "paints" it over top of the original. This provides more visual feedback when moving the mouse over a link area which is superior to guessing where to click and hoping you're right. Figure 3.1 shows how the applet works.

Figure 3.1 Theory of operation



Starting Out

Before you begin, locate the ImageMap Tutorial folder. It is in the following location: `CodeWarrior Examples\CodeWarrior Java\Metrowerks Sample Projects\Java Tutorial\Im-`

ageMap Tutorial. If this folder is not already installed on your hard drive, copy it from the CodeWarrior Reference CD to your hard drive. The files you create and modify during this tutorial will be located in this folder.

All of the files for the completed tutorial can be found in the `ImageMap Solution` folder. You can compare your work against those files if you run into problems.

WARNING! You should be very careful to use the exact names specified in each step of this tutorial. Some features of Java are very dependent on names, and the applet may not work if you change names. Java is also a case-sensitive language, so make sure you match case as well as spelling.

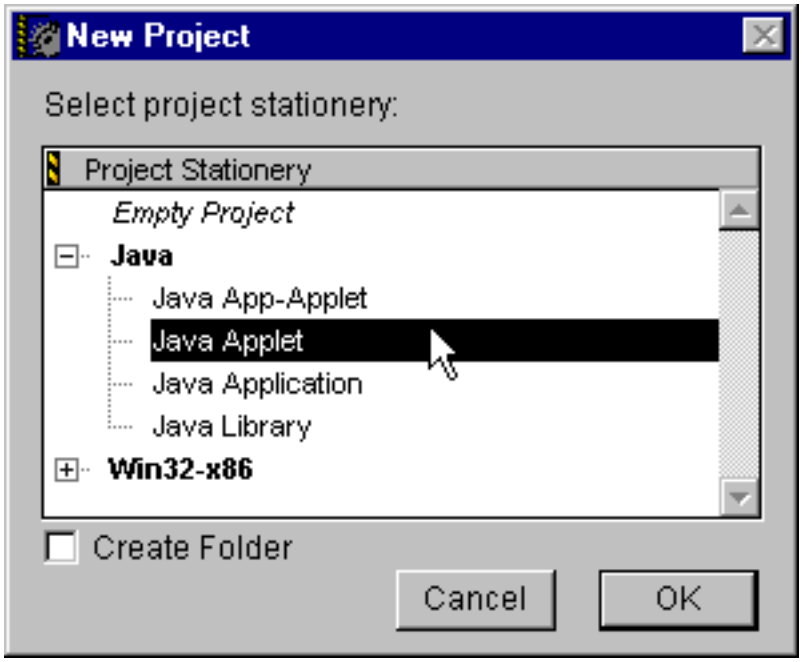
Let's get started. The topics covered in this section of the tutorial are:

- Creating a project
- Changing Target Settings

Creating a project

Locate the CodeWarrior IDE application and launch it. After it opens, choose **New Project** from the File menu. The New Project dialog box appears (Figure 3.2).

Figure 3.2 The New Project dialog box

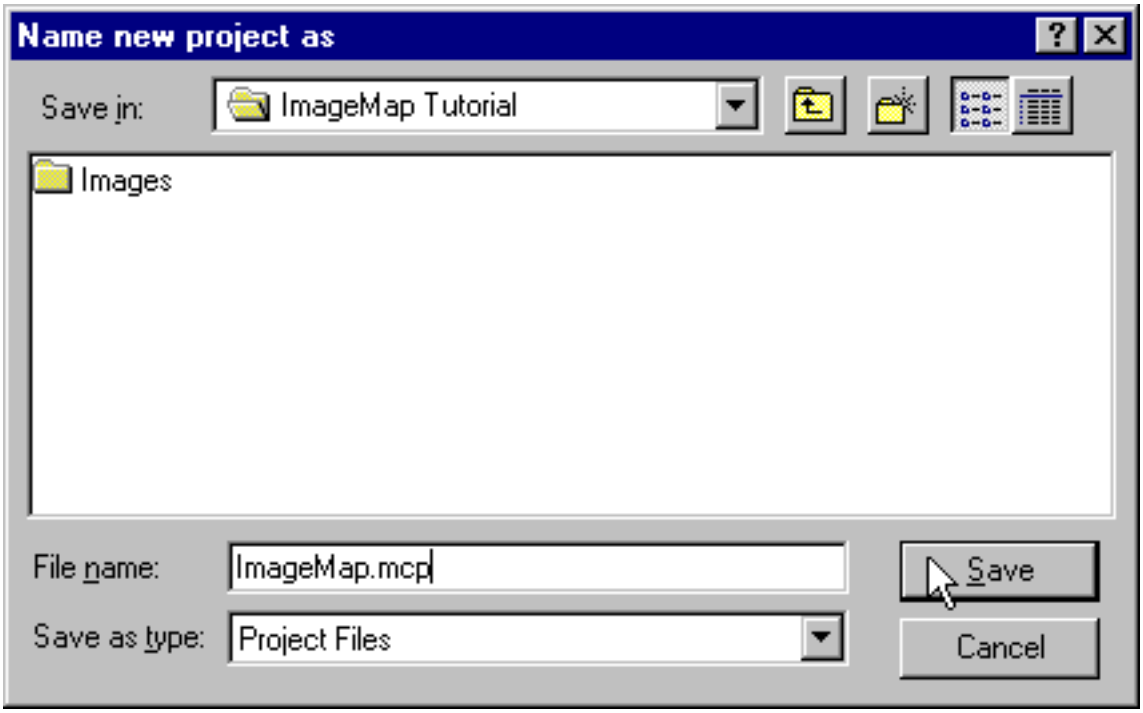


Select the Java Applet stationery. If the stationery isn't visible, click the tree control beside the Java group to make it visible, then select it. Also make sure the **Create Folder** check box is *off*. Then click **OK**.

CodeWarrior now displays a save file dialog. Navigate to the `ImageMap Tutorial` folder on your hard drive. The dialog should look like the one in Figure 3.3. Name your project `ImageMap.mcp`.

NOTE: The "`.mcp`" suffix is a CodeWarrior naming convention for project files. If you do not add "`.mcp`", CodeWarrior will add it for you.

Figure 3.3 The name new project dialog box



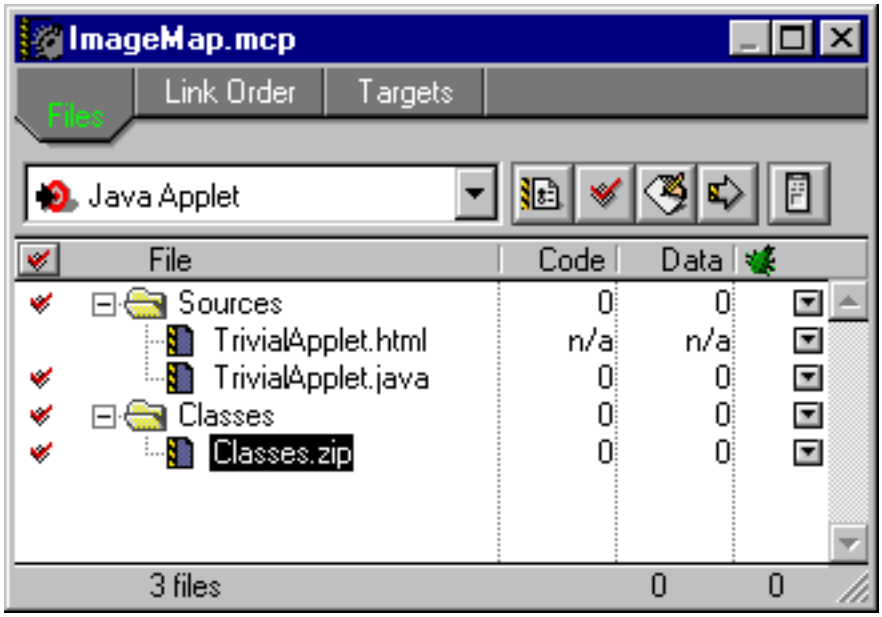
Click the **Save** button. CodeWarrior creates three files in the tutorial folder:

- The project file `ImageMap.mcp`
- The HTML file `TrivialApplet.html`
- The Java file `TrivialApplet.java`

CodeWarrior also creates a "data" folder. This folder contains files with information about your project file, target settings you've chosen, object code, and browser information. You should not change the contents of this folder.

Code Warrior opens the project and displays a window similar to that shown in Figure 3.4. You've got a project under way!

Figure 3.4 The project window



Changing Target Settings

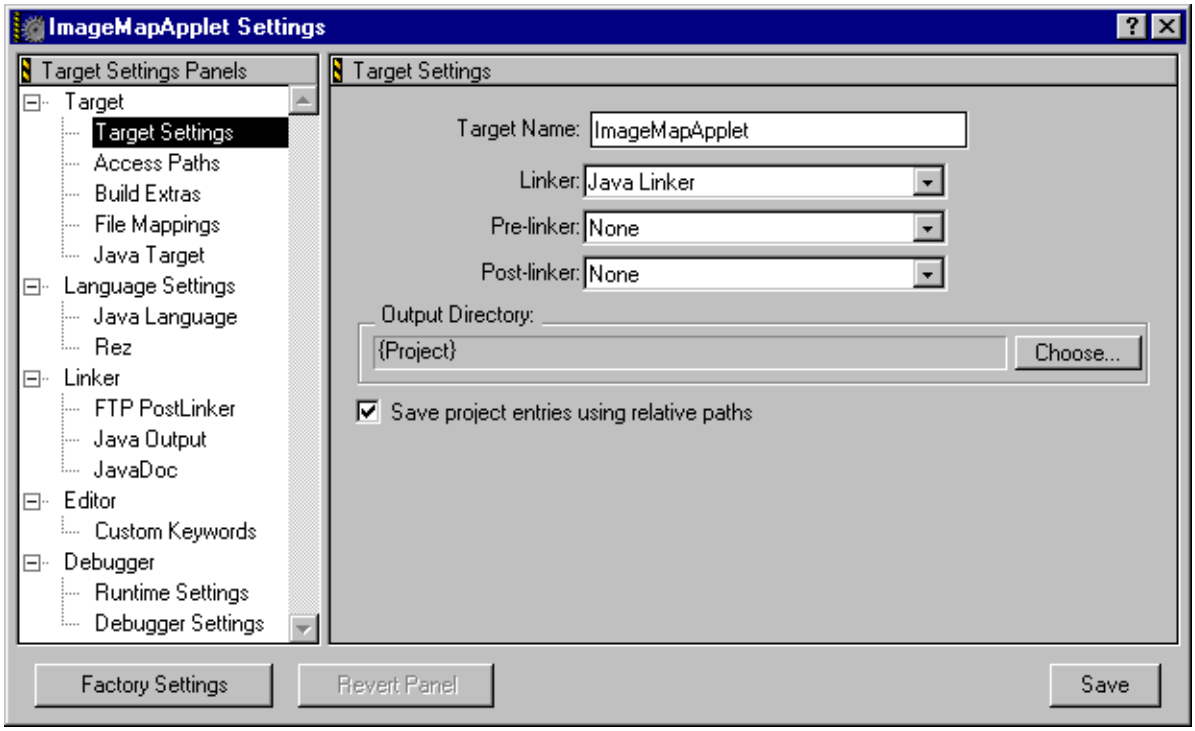
A project file can have multiple build targets. For example, you can have a *debug* build target and a *release* build target in the same project. Each build target has its own unique collection of settings that control how CodeWarrior creates the final executable. These are the build target settings. When you change the settings for a build target, the change affects only the current build target.

NOTE: CodeWarrior also has global preferences that affect all projects. For more information on preferences and target settings, see the *IDE User Guide*.

By creating a new project using stationery, the default build target settings are already set for you. However, for most projects, you will need to change a few of these settings in order to get the results you are looking for.

Choose ***MyTarget* Settings** from the Edit menu, where ***MyTarget*** is the name of the currently active build target in the front-most project. In this case, the build target name should be **Java Applet**. CodeWarrior displays the Target Settings dialog box (Figure 3.5).

Figure 3.5 Target settings dialog box



The Target Settings dialog box contains a list of *settings panels* on the left. The right side of the dialog box shows the options for the currently selected panel.

For the purposes of this tutorial, we're only interested in changing a few options in three of these panels. For a more detailed description of each settings panel, see "Target Settings for Java" on page 73.

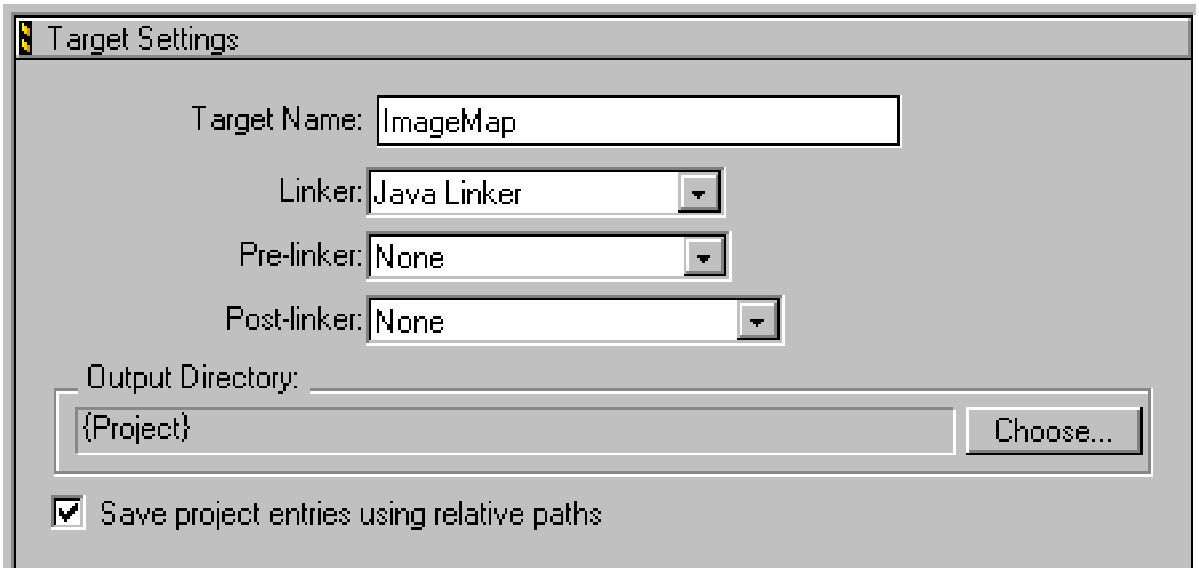
Changing target information

Here, you change three settings for this target: the name, the applet viewer, and the output. These settings are in three different panels.

1. **Change the target name.**

   Click **Target Settings** from the list on the left. Change the **Target Name** field to "`ImageMap`" (Figure 3.6).

Figure 3.6 Target Settings panel

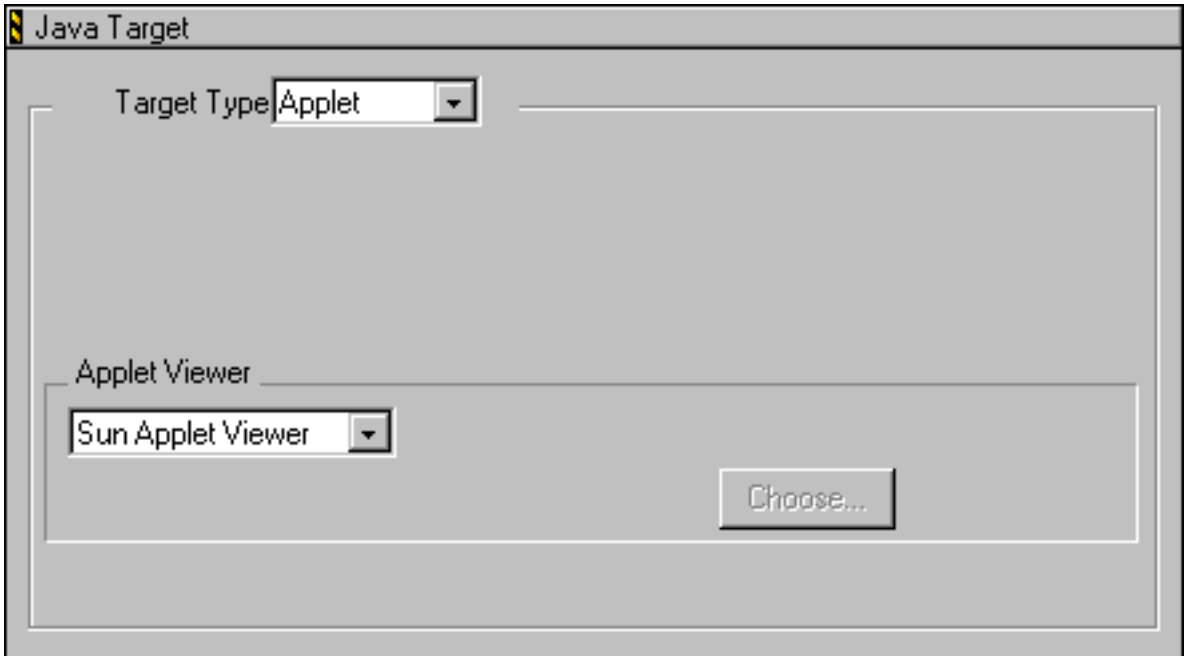


2. **Change applet viewer.**

   Click **Java Target** from the list on the left. Depending on which you have installed (or which one you prefer), set **Applet Viewer** to be Internet Explorer or Suns Applet Viewer by selecting it from the popup menu. The panel should look like Figure 3.7.

   You may also choose to use another application to view your applet by selecting **Other** fromt he popup menu, and then clicking the **Choose** button, and finding the application on your hard drive. However, if you do so, you will not be able to use the CodeWarrior Debugger to debug your project.

Figure 3.7 Java Target panel

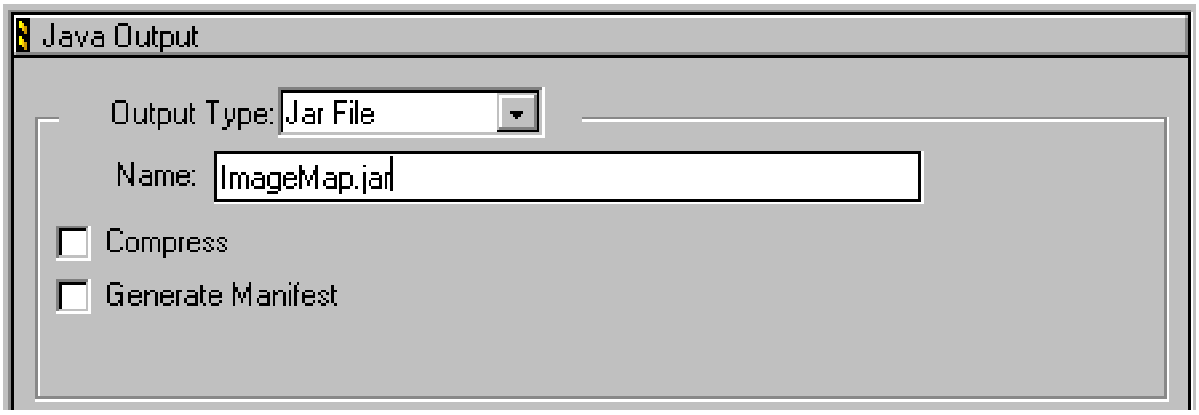


3. **Change the output.**

   The only other settings panel we are interested in is the **Java Output** panel in the Linker section. Select the **Output** panel in the list of settings panels.

   The **Output** panel controls the type and name of the file CodeWarrior creates when it links your code. Make sure **Type** is set to **Jar File** and the **Name** field is set to `ImageMap.jar` (Figure 3.8). These settings create a new Java Jar archive in same folder as your project.

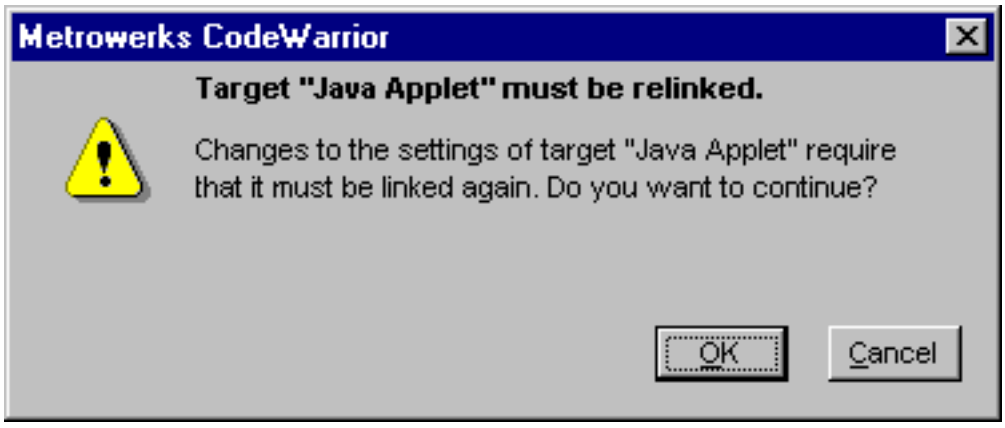
Figure 3.8 Output panel



Now that all the settings have been changed, you're ready to start writing the code. Click the **Save** button at the bottom of the Target Settings dialog box. Close the Target Settings dialog box.

NOTE: Depending on which options were changed, you may see the following dialog upon closing the dialog. Clicking the Cancel button will cancel the closing of the dialog and allow you to make more changes. Clicking the OK button will close the dialog.

Figure 3.9 Relink applet dialog


Metrowerks CodeWarrior
Target "Java Applet" must be relinked.
Changes to the settings of target "Java Applet" require that it must be linked again. Do you want to continue?
OK
Cancel

Writing the Applet

In this section, you write the actual code for the applet. Topics discussed in this section of the tutorial are:

- Adding the Java File
- Editing the HTML File

Adding the Java File

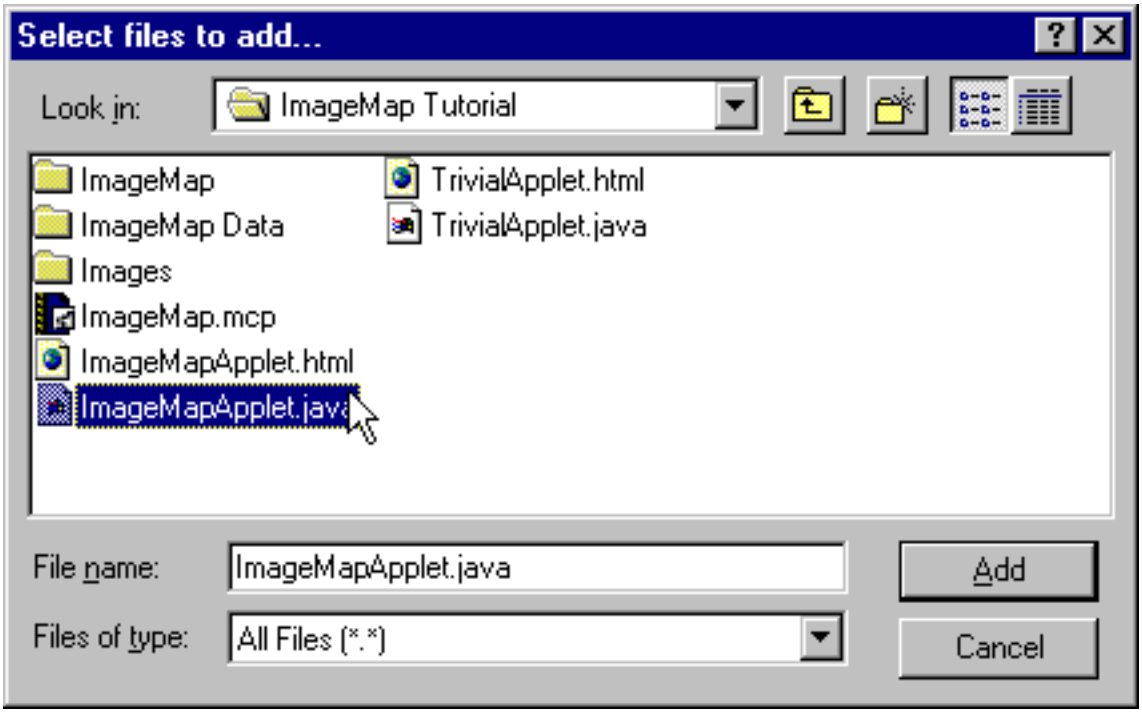
The code for the ImageMap applet is extensive at almost 200 lines. To make things easier, the code is provided for you in the tutorial folder. You add this file to the project.

1. **Add ImageMapApplet.java to the project.**

   To make sure your file is added to the appropriate section of the project window (in this case, in the Sources folder), first highlight any file in that section or the section title itself. Then choose **Add Files** from the Project menu. The Add Files dialog box shown in Figure 3.10 will appear.

   Select `ImageMapApplet.java` and click the **Add** button as shown in Figure 3.10.

Figure 3.10 Add files dialog box



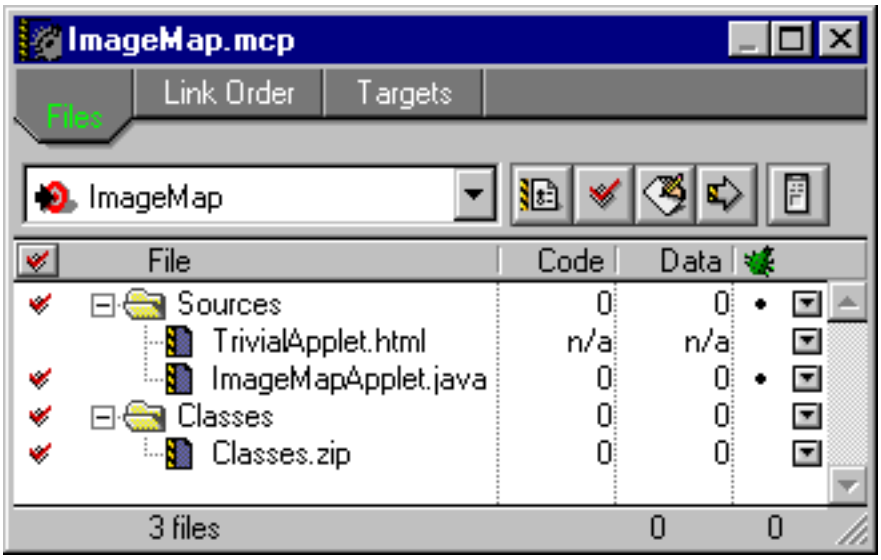
2. **Remove placeholder file.**

   Now, remove the `TrivialApplet.java` file from the project by selecting it, and then choosing **Remove Selected Items** from the Project menu. The resulting project window should look like Figure 3.11.

TIP: For a faster way to remove items in the project window, select the file or item you wish to remove. Hold down the Control key and press Delete.

The file `TrivialApplet.java` is still in the `ImageMap Tutorial` folder. You can delete it if you want, but leaving it there will not affect your project.

Figure 3.11 Java code added to the project



Examining the code

Open the `ImageMapApplet.java` file and look at the source code.

NOTE: Do not make any changes to the source code yet.

Reading the comments in the code will help you understand the functionality of the applet. If you aren't familiar with Java syntax, you may not understand all the code. That's OK. The purpose of this tutorial is not to teach you Java, but to introduce you to CodeWarrior. See "Where to Go from Here" on page 9 for references to Java language documentation.

The applet uses the Java API to display a picture. When you move the cursor over certain defined areas of the picture, that area changes. The images used can be found in the `Images` folder.

Java naming conventions

The name of your Java file is important. The name of the file must be the same as the name of the class created in the file, and end with the `.java` extension. This extension is a Java programming conven-

tion for naming source files. The Metrowerks Java compiler uses this extension to recognize Java source files more easily.

NOTE: Actually, the Metrowerks Java compiler lets you name a file whatever you want. This feature is useful on the Mac OS, which limits file names to 32 characters: too short for some Java class names. However, you should name a Java file after its class whenever possible, so your code is compatible with all Java compilers. If you must give a file a different name, choose a name that's as close as possible to the class's name.

You have all the code for a complete, interactive Java applet. You're most of the way there. There are just two more things to do before you test the applet.

Add Graphics Files

Part of the flexibility in using a Jar file for the applet is that it can contain files that are not Java Classes. For this project, we want to include the two graphic files used for the image map in the archive.

There are two ways to accomplish this. You can add the graphics manually by editing the Jar file with a ZIP utility, or you can add them directly to the CodeWarrior project file to be merged into the final jar file when you build the project.

In order to add these graphics directly to the CodeWarrior project, there are two things that need to be done. First, the current target in the project must be able to recognize the graphic file. Second, after the files are added, you need to tell CodeWarrior to merge the graphics into the Jar file.

Recognizing graphics files

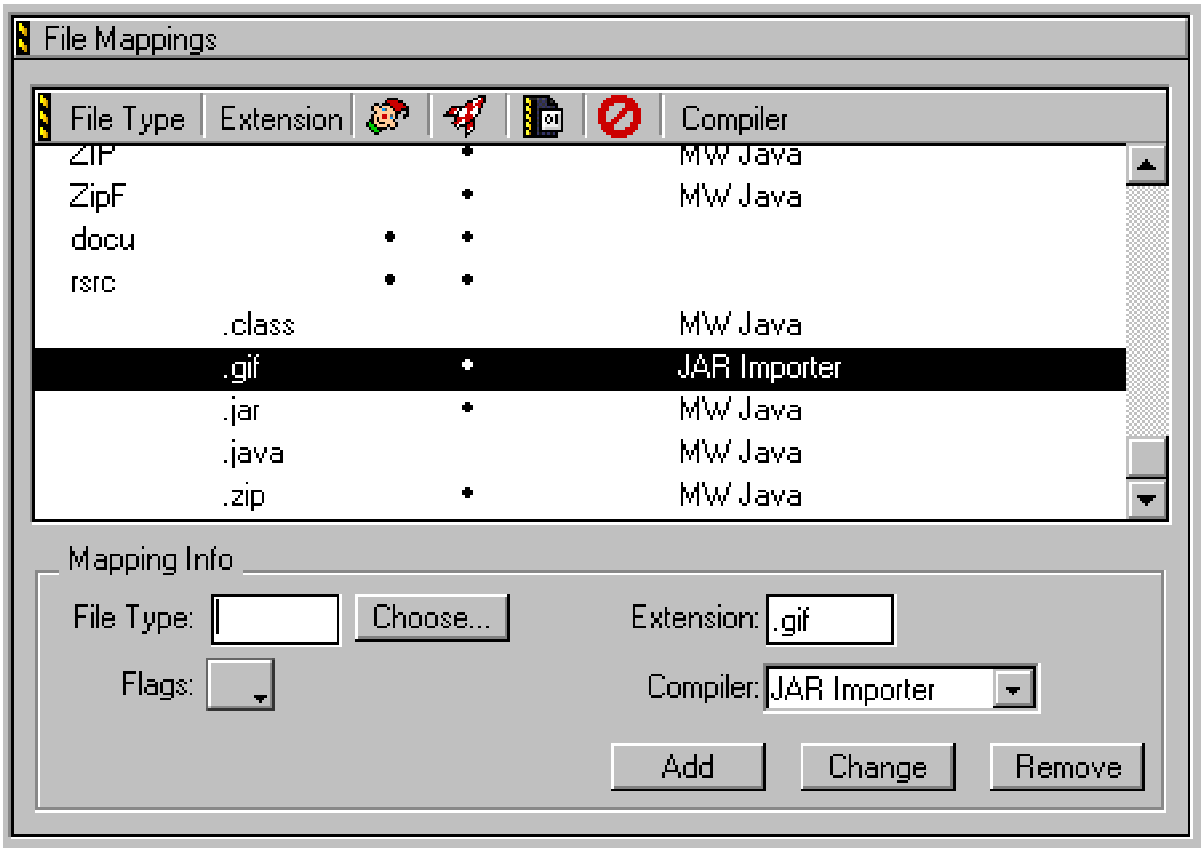
The **File Mappings** settings panel allows you to add, change, or remove file types recognized by your build target by using a character extension added to the file name, such as `.gif`, `.jpg`, or `.html`. The default file mappings are set for every new target in a

CodeWarrior project file. You can change these settings on a per target basis.

Choose ***ImageMap*** **Settings** from the Edit menu (remember, ***ImageMap*** is the name of the currently active target in your project). Choose File Mappings from the list of settings panels on the left. Look over the list of file mappings. Make sure there is an entry for `.gif` as shown in Figure 3.12. If there isn't an entry, type in the data as shown. Make sure the Launchable flag is the only one with a check mark in the **Flags** popup menu. When you have entered all of the necessary data, click the **Add** button.

TIP: Setting the Launchable flag allows CodeWarrior to launch your favorite application that handles that file type when you double-click on it in the Project window.

Figure 3.12 File Mappings panel



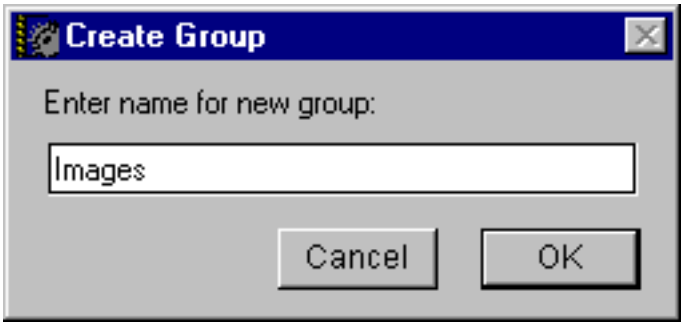
Setting up the graphic files

Now that the file mappings are set to recognize the two graphic files, they can be added to the project. However, it is important to add the graphics to a new group that mirrors their physical directory structure.

1. **Create new group**

   Choose **Create New Group** from the Project menu. Name the group "Images" as shown in Figure 3.13. Click **OK**.

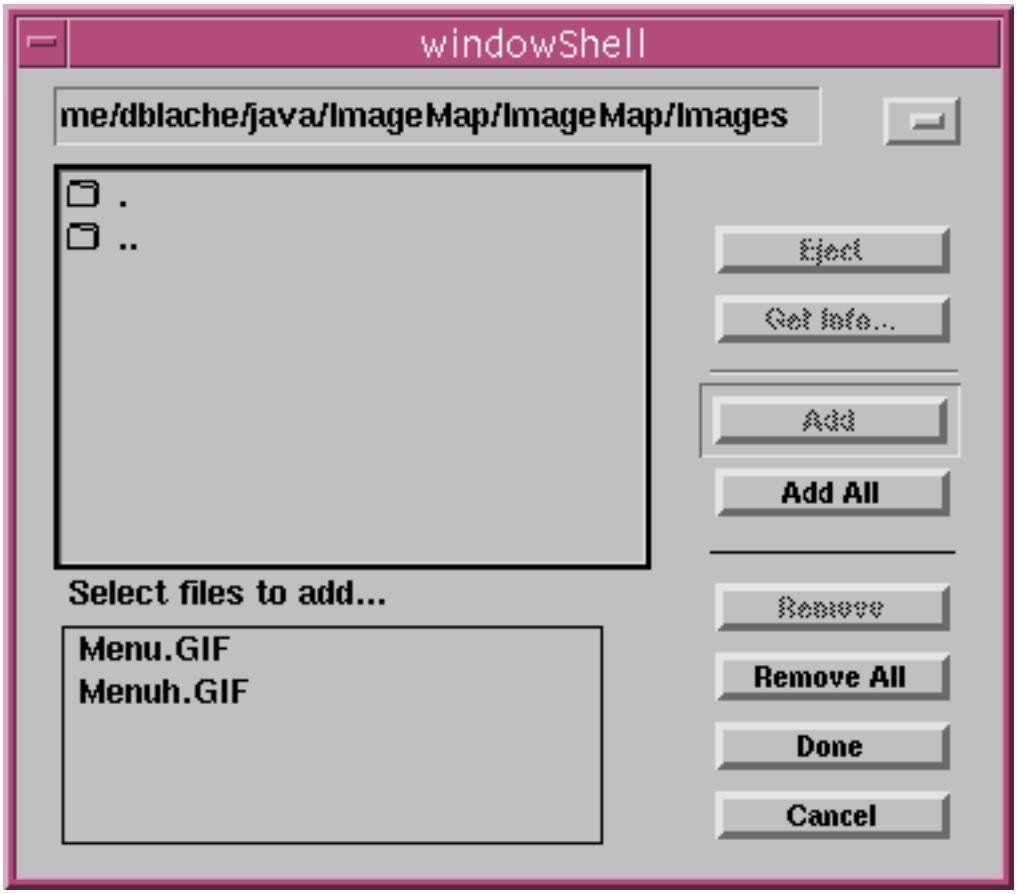
Figure 3.13 New group dialog

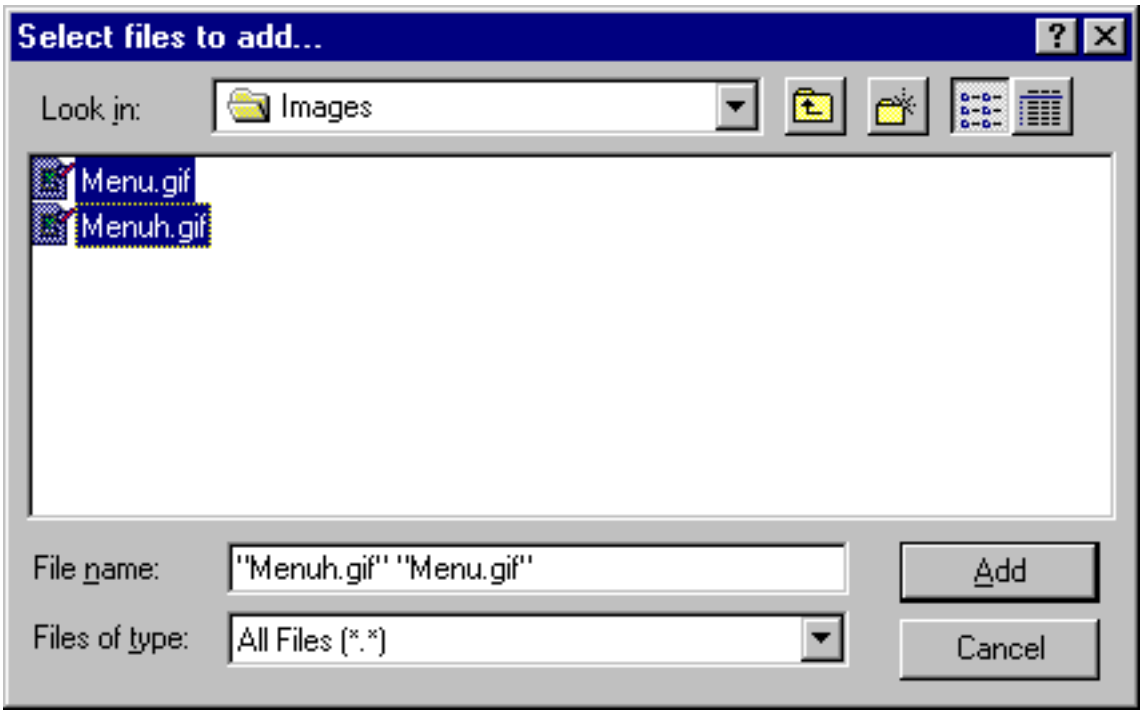


2. **Add menu graphics to the new group.**

   Click on the Images group in the Project window to select it. Choose **Add Files** from the Project menu and add both graphic files (Figure 3.14). The files will be added to the Images group.

Figure 3.14 Adding graphics

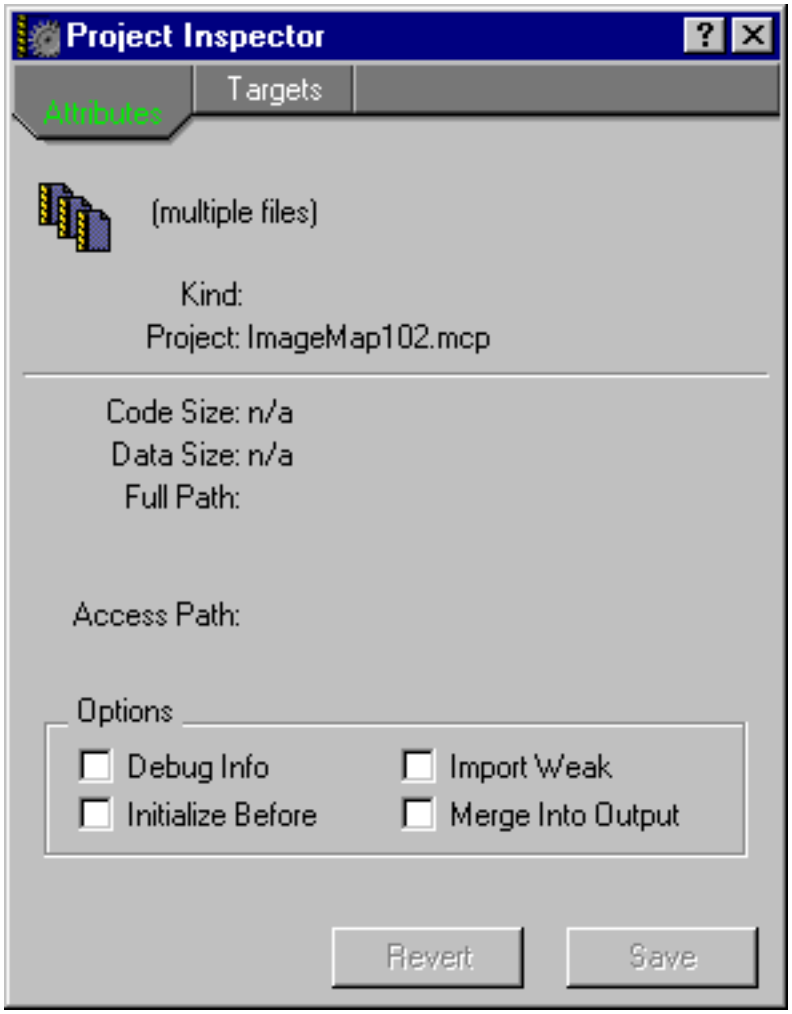




3. Set merge flag

To select both graphics files, hold the Shift key down and click once on each graphics file in the Project Window. Choose Project Inspector from the Window menu. Click the Merge Into Output checkbox and click **OK** (Figure 3.15). Close the window.

Figure 3.15 Merge setting



Now, when the applet is compiled, the graphic images will be added directly into the Jar file under the heading "Images."

Editing the HTML File

In this section, you edit `TrivialApplet.html` and rename it. The HTML file is used by the web browser to pass parameters to your Java applet.

CodeWarrior allows you to add non-code files to a project. In this way you can keep track of all files related to a project, even those that are not compiled. The project stationery for an applet project includes a sample file, `TrivialApplet.html`.

When developing an applet, you typically want to have a small HTML file that embeds the applet code. CodeWarrior uses this file to run your program from within the IDE. When you choose **Run** from the Project menu, CodeWarrior opens the first HTML file in the **Link Order** view of your project. The order of HTML files in the **Files** view of your project does not matter.

For more information on either the File view or Link Order view of the CodeWarrior project window, see the *IDE User Guide*.

1. **Edit TrivialApplet.html**

   Double-click `TrivialApplet.html` in the project window to open it. `TrivialApplet.html` is a simple HTML file that displays an applet and gives you a link to its source code. Delete the code in `TrivialApplet.html` and replace it with code shown in Listing 3.1.

TIP: If you don't feel like typing in the code, you add it to the project using the **Add Files** command. The `ImageMap.html` file is supplied in the `ImageMap Tutorial` folder for this purpose.

Listing 3.1 The HTML file for ImageMapApplet

```
<html>
<head>
<title>Java Tutorial</title>
</head>
<body bgcolor="#CC9966" text="#000000" link="#0000FF"
vlink="#000080" alink="#FF0000">
<basefont size=2>
<div align=left>
<applet archive="ImageMap.jar" code="ImageMapApplet.class"
width=248 height=172>
<param name="normal" value="images/menu.gif">
```

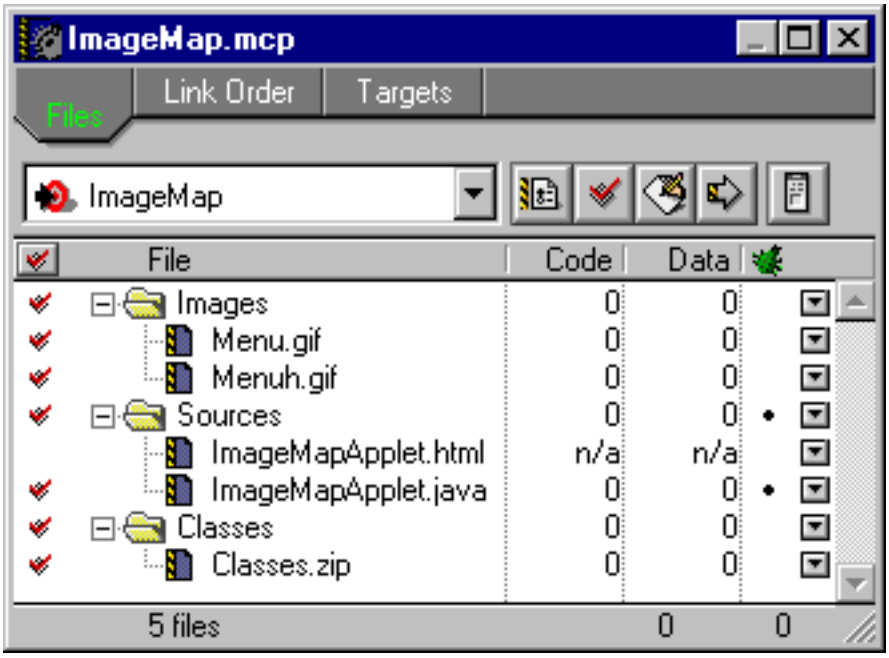
```
<param name="highlighted" value="images/menuh.gif">
<param name="URL0" value="http://www.metrowerks.com/">
<param name="URL1" value="http://www.gamelan.com/">
<param name="URL2" value="http://search.yahoo.com/bin/
search?p=java">
<param name="URL3" value="http://java.sun.com/docs/index.html">
</applet>
</body>
</html>
```

The HTML code tells the Web browser that the applet code can be found in a file named `ImageMap.jar`. It also sets the viewing size of the applet to be 278 by 172 pixels.

2. **Save the file under a new name**

   Choose **Save As** from the File menu. Make sure you're in the `ImageMap Tutorial` folder. Name the file `ImageMapApplet.html`. CodeWarrior saves the file with the new name and changes the name of the file in the project window, as shown in Figure 3.16.

Figure 3.16 The project window with the renamed HTML file.



You've removed the `TrivialApplet.html` file from your project; but it is still in the `ImageMap Tutorial` folder. You can delete it if you want, but leaving it there doesn't affect your project.

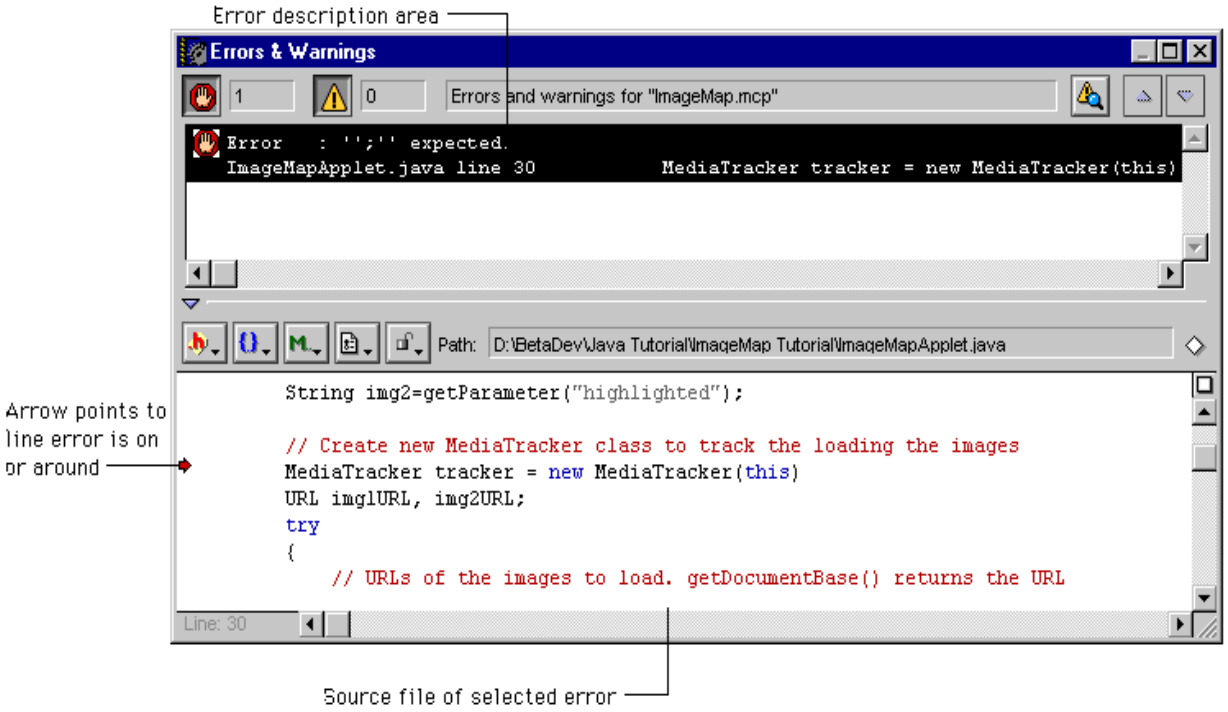
Compile and Run

You are now near the final stages of your first Java applet, and this tutorial. All that is left to do is compile and test it to make sure it runs smoothly.

Choose **Make** from the Project menu. The **Make** command compiles and links your code into the finished applet, all in one step.

NOTE: The CodeWarrior IDE is very smart about picking up syntax errors at compile time. Usually these errors are the result of mistyped or unused variable names, incorrect class information, or other related errors. The tutorial code has one deliberate error (Figure 3.17).

Figure 3.17 Compiler errors


Error description area
Errors & Warnings
1
0
Errors and warnings for "ImageMap.mcp"
Error : ''';'' expected.
ImageMapApplet.java line 30 MediaTracker tracker = new MediaTracker(this)
Path: D:\BetaDev\Java Tutorial\ImageMap Tutorial\ImageMapApplet.java
Arrow points to line error is on or around
String img2=getParameter("highlighted");
// Create new MediaTracker class to track the loading the images
MediaTracker tracker = new MediaTracker(this)
URL img1URL, img2URL;
try
{
// URLs of the images to load. getDocumentBase() returns the URL
Line: 30
Source file of selected error

The Errors & Warnings window displays a list of errors the compiler found when it tried to make the project, in this case only one. The bottom part of this window shows the source file that contains the selected error. The red arrow points to the specific line of code where the error was encountered. Sometimes the error is on the same line, but it can also be found a line or two before depending on the type of error.

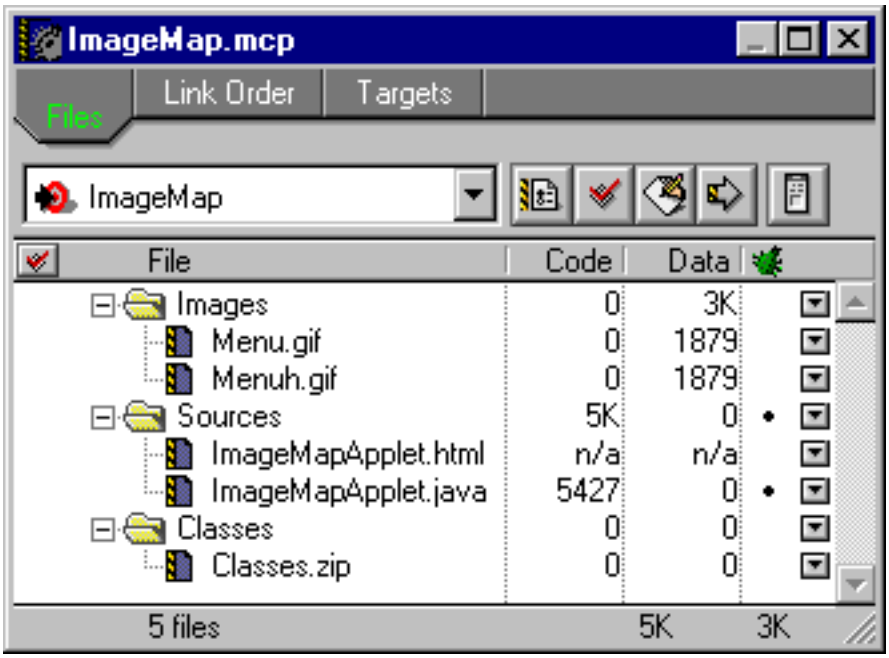
The source view of the Errors & Warnings window is completely editable. You can edit and save your code directly from this window. There is no need to open up the source file and try to find the error manually. This is a great time saver if you have many errors.

In this case, the error is a missing semicolon (;) at the end of the line pointed out by the red arrow. Click at the end of the line and type a semicolon (;). Then choose **Save** from the File menu to save the file. Close the Errors & Warnings window, then choose **Make** from the Project menu to compile the project again.

While building your project, the check mark to the left of the files in the project window disappears, indicating that the files no longer need to be compiled. In the Code column, the number of bytes of code is updated. The number is zero for the `classes.zip` file because that is a shared library to which the applet links at runtime. This file contains the Java classes and methods called from the `ImageMapApplet.java` source file.

Your project is completely compiled and ready to run (Figure 3.18).

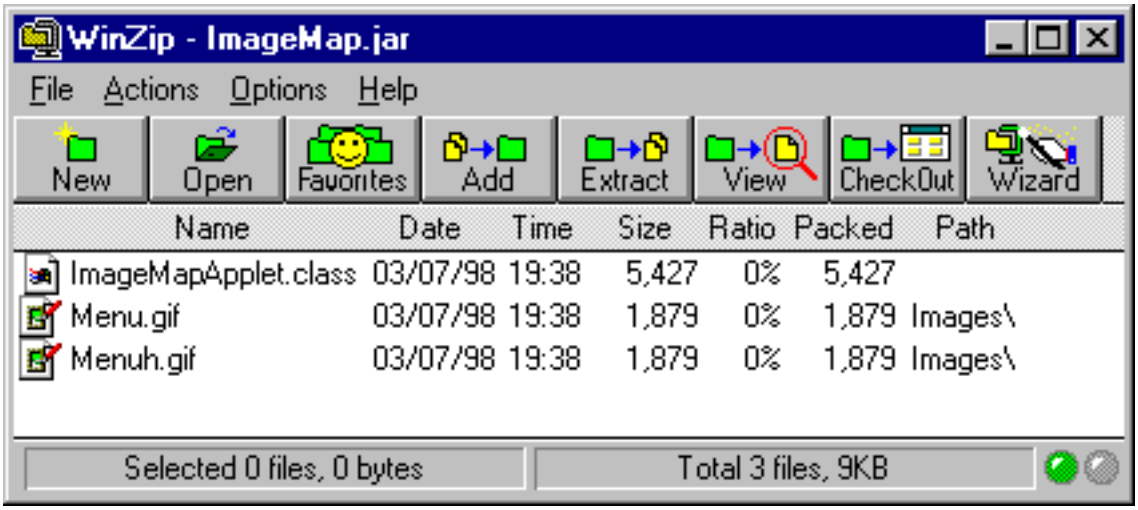
Figure 3.18 Compiled java project



Examining the output

Take a minute to examine the Jar file produced by CodeWarrior with your favorite ZIP tool. It should look similar to Figure 3.19. Notice that the graphic files have been merged into the file in a directory structure matching the name of the group they were under in the project window. Notice also that the Jar file is not compressed. Do not compress the file! While you save disk space using a compressed Jar, the applet will run very slowly.

Figure 3.19 Jar file


WinZip - ImageMap.jar
File Actions Options Help
New Open Favorites Add Extract View CheckOut Wizard
Name Date Time Size Ratio Packed Path
ImageMapApplet.class 03/07/98 19:38 5,427 0% 5,427
Menu.gif 03/07/98 19:38 1,879 0% 1,879 Images\
Menuh.gif 03/07/98 19:38 1,879 0% 1,879 Images\
Selected 0 files, 0 bytes
Total 3 files, 9KB

Run the applet

The last thing to do at this point is test whether the applet works. Choose **Run** from the Project menu. This launches Internet Explorer using the HTML file in your project. The HTML file, in turn, tells Internet Explorer where to find the class file and graphics used in the applet.

NOTE: If you are connected to the Internet via modem, either with SLIP or PPP, your applet viewer may try to start a connection. You do not have to be connected to the Internet for this part of the tutorial, though you can be if you wish. If you don't want to be connected, just cancel the connection. The applet will continue to work without it.

Do not make any selections in the applet at this time. Move the cursor over each word in the graphic. The word changes color from blue to yellow, indicating the current selection. However, there is a bug in the applet.

Oops! What's Happening?

As you pass the cursor over "Gamelan," the word changes to show a half highlighted state and the text is displaced (Figure 3.20). The bug that's causing this problem could be in one of several places in the code. Now, you have to debug the applet to find out what is going on.

Figure 3.20 Applet bug

There is one primary topic discussed in this section:

- Using the CodeWarrior Debugger

Using the CodeWarrior Debugger

The CodeWarrior Debugger is integrated into CodeWarrior to help you debug your code. The same debugger works for Java, C, C++, Pascal, and assembly language.

The debugger is called a "source level debugger" because it displays your source code so you can see exactly which line will be executed next. This has the added benefit of getting a clearer understanding of your code, without the need to learn assembly language.

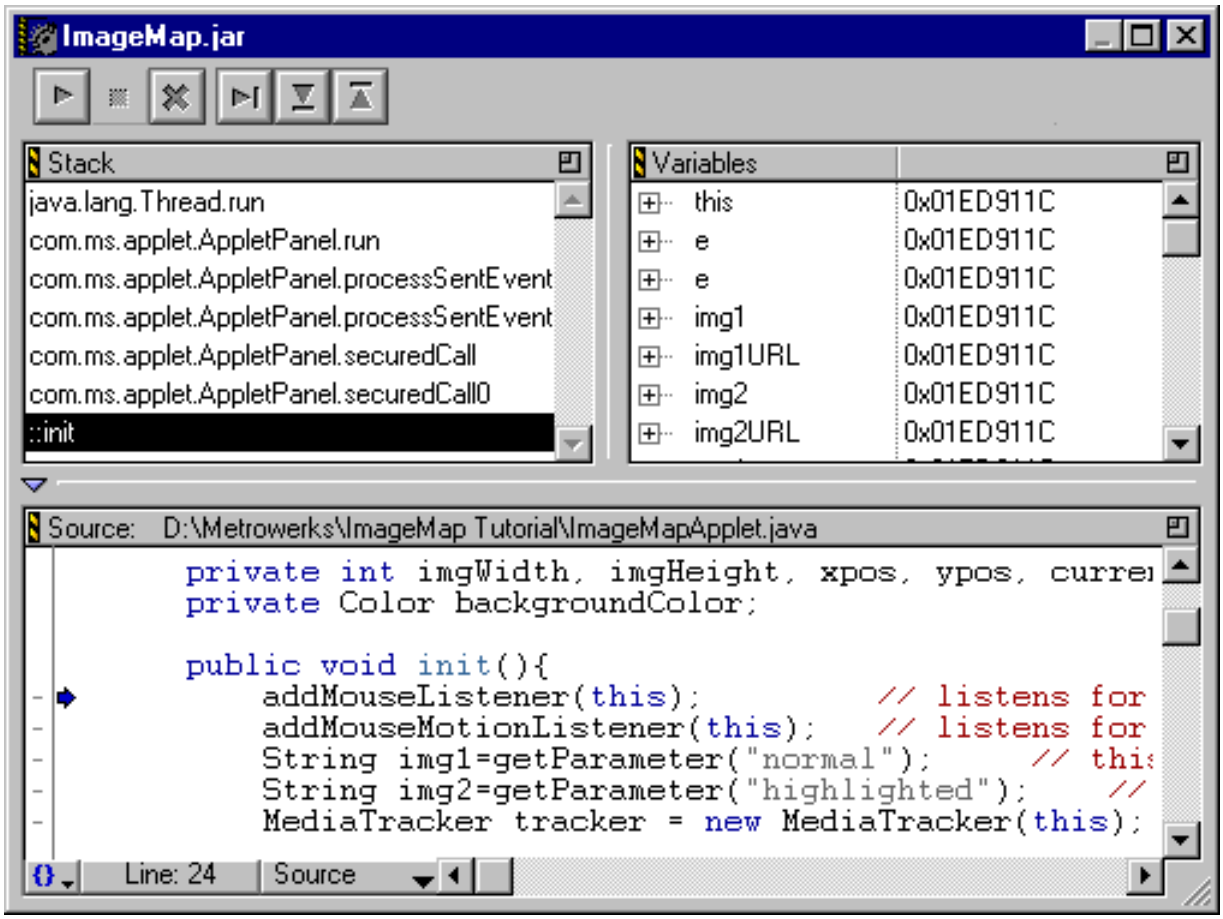
For more information on the debugger, see the *Debugger User Guide*.

Debugging the applet

Before you can debug your Java code, you must first tell CodeWarrior to use the debugger when running the applet. Choose **Enable Debugger** from the Project menu. Now you are ready to start a debugging session. Choose **Debug** from the Project menu. CodeWarrior relinks the project and opens the debugger.

CodeWarrior opens the debugger and passes the first HTML file in the Link Order view of your project to the debugger. The debugger launches Internet Explorer which loads the applet and creates a window for it to run in. Finally, the debugger comes back to the front, displaying the program window for the applet, with the current statement arrow at the first line of code in the `init()` method, as shown in Figure 3.21.

Figure 3.21 The debugger's program window



The debugger allows you to examine local variables, step through the program, set breakpoints, and perform other debugger functions. The top left pane is the Stack Crawl Pane. The Stack Crawl pane shows the order in which each method has been executed, called a *call chain*. The arrow in the bottom pane is called the *current statement arrow*. This arrow points to the line of code that will be executed next. See the *Debugger User Guide* for a more detailed description of the debugger windows.

Solving the Problem

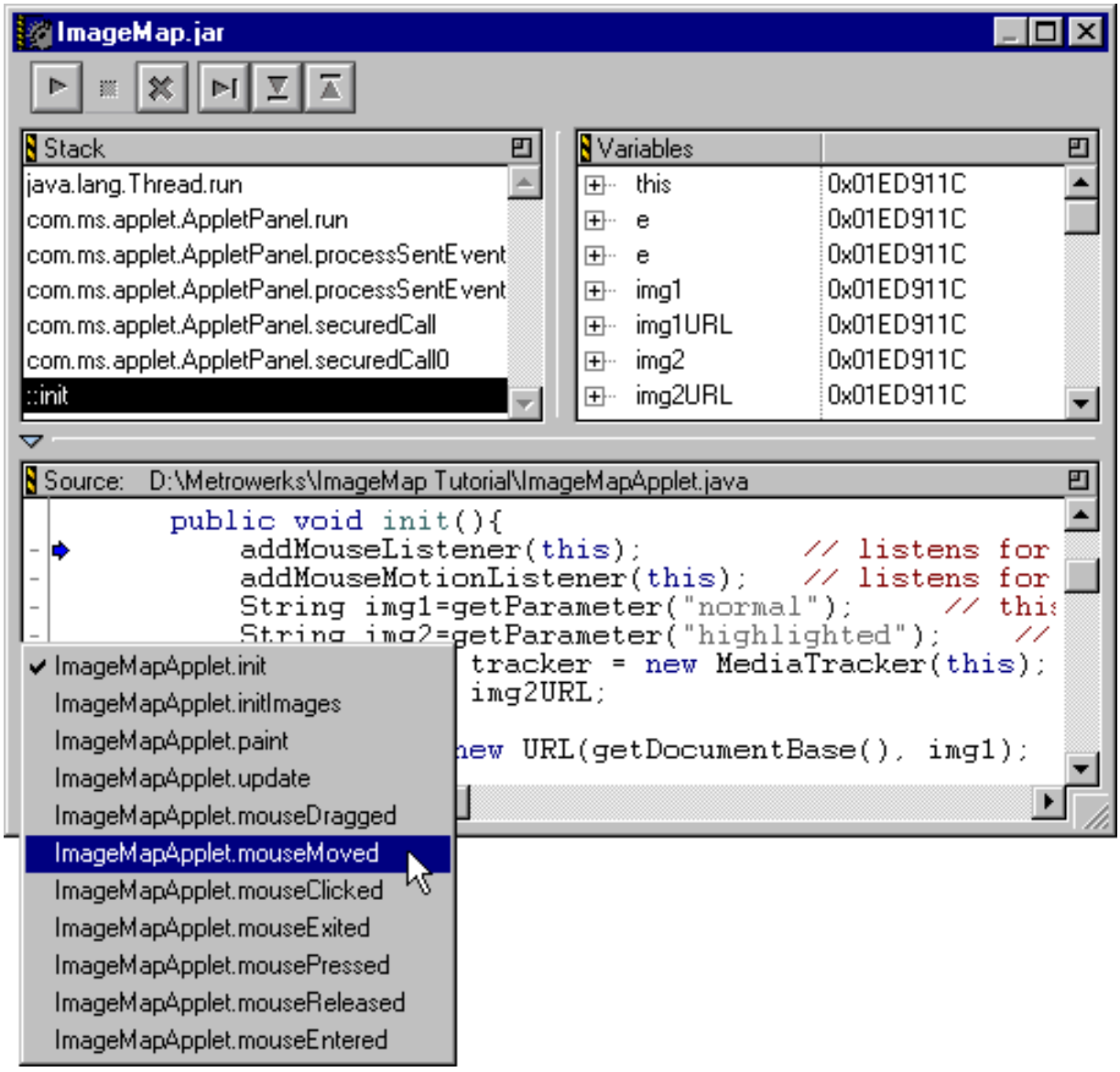
To solve the bug in this applet, it's necessary to review what happened and when. This gives you greater understanding of the code and a better idea where to start looking for the problem.

Remember that the applet essentially seems to run correctly with the exception for one item. As the mouse moved over the menu graphic, it changed to display an incorrect image. That's where to start looking, the `mouseMove()` method.

1. **Select the `mouseMove()` method in the debugger.**

   The bottom left corner of the debugger window contains a small icon. This icon is a pop-up menu. Click on the pop-up menu and choose the `mouseMove()` method (Figure 3.22). The debugger scrolls the source code to show the first few lines of the selected method.

Figure 3.22 Choosing a method

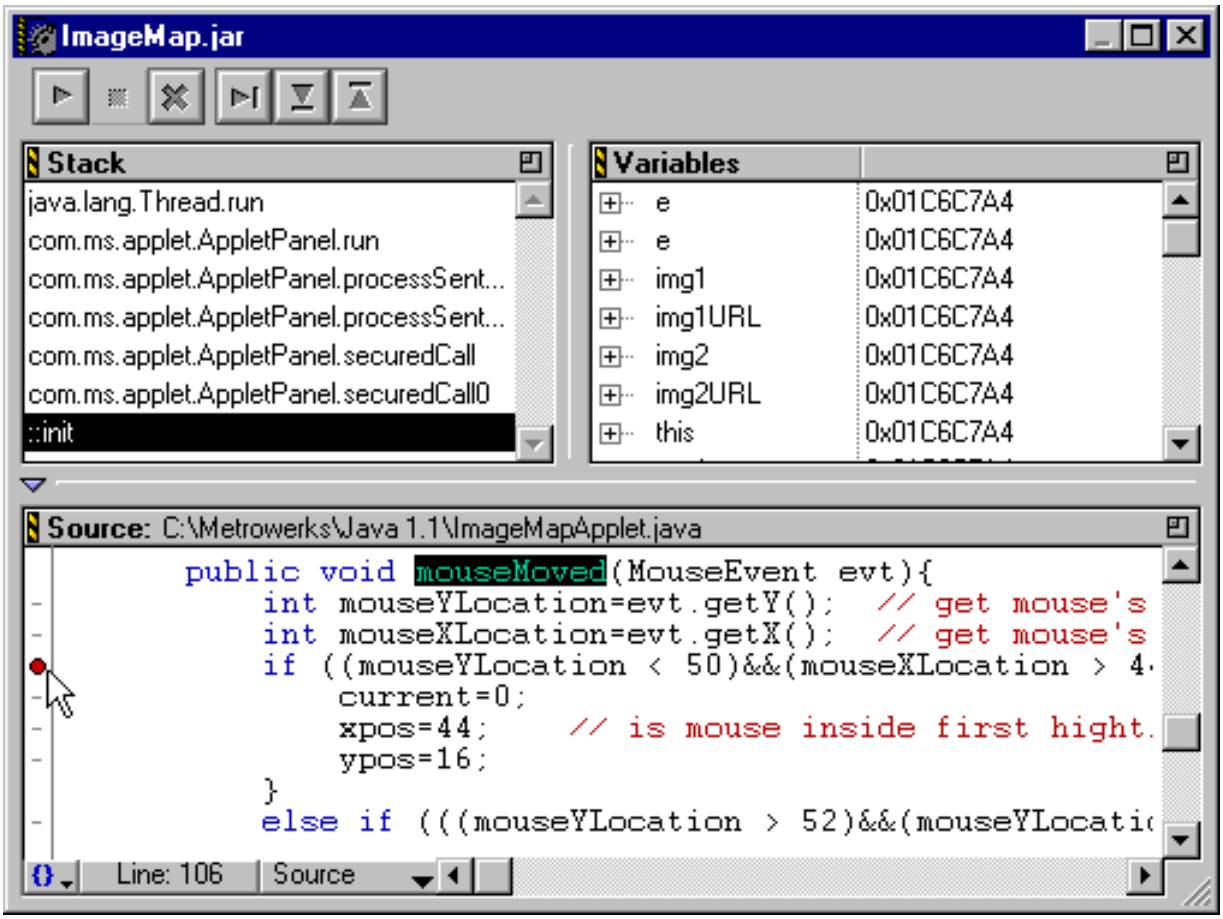


A *breakpoint* is like a stop sign. It tells the debugger to execute all code up to the line just before the breakpoint. Each dash (-) symbol

in the left column indicates a line of code where a breakpoint can be set.

You want to set a breakpoint at the first available line of code in the `mouseMove()` method. To set the breakpoint, simply click the dash symbol, as shown in Figure 3.23.

Figure 3.23 Setting the breakpoint



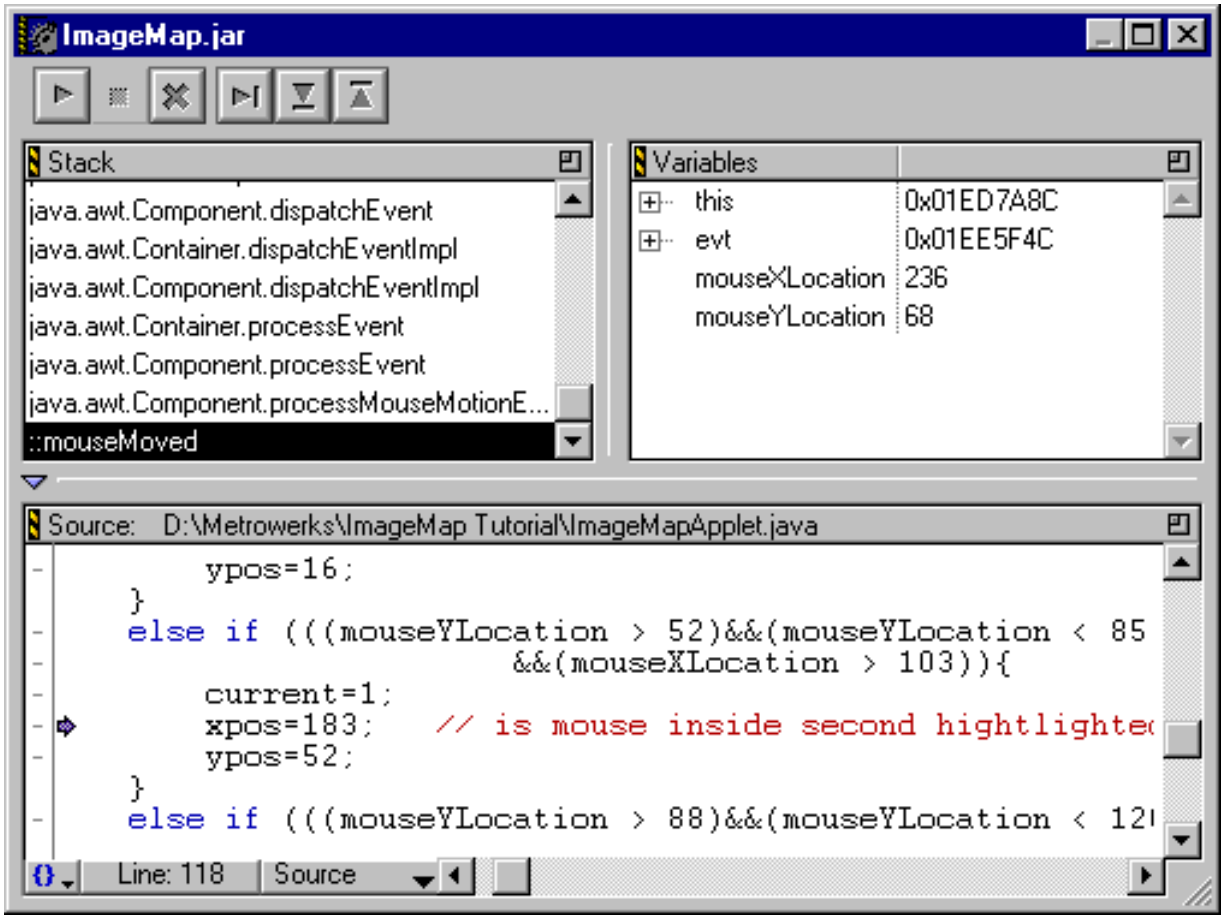
Choose **Run** from the Project menu. This allows Internet Explorer to run the applet until it reaches the line of code where you set the breakpoint. It won't reach that line of code until you move the cursor over the applet.

Move the cursor over the second item in the list of the graphic. The debugger becomes the front-most application and the current statement arrow is on the same line as the breakpoint. At this point, you

want to examine each line of code as it executes to try to find the error. This process is called *stepping* through your code.

Choose **Step** from the Control menu. The debugger executes that line of code. Continue to use the **Step** command until you reach the line of code shown in Figure 3.24.

Figure 3.24 Offending line of code



This is the culprit! This line tells the applet to act as if the cursor is at a position of 183 pixels on the X axis. Really, this should be 103! Small errors like this are common when typing code and can cause unnecessary headaches in your own applications.

Editing the source file

The debugger is not an editor. You cannot edit your code directly in the debugger window. Halt execution of the applet by choosing Kill from the Debug menu. Then open the file by double-clicking it in the project window.

Change the line of code as shown in Listing 3.2.

Listing 3.2 Corrected java code

```
xpos = 103;
```

Recompile and run

Now that you've changed some code, it's necessary to recompile the applet and run it again. This time, you run it without the debugger.

Choose **Disable Debugger** from the Project menu. Choose **Run** from the Project menu. If you have an internet connection, let your web browser start the connection. Now move the cursor over the graphical menu. Try clicking on each item in the menu to see where it leads. **Congratulations**! You've completed your first Java Applet!

Exercises

There is still more that can be done with this applet. Currently, clicking on a navigational link brings up a second browser window. It would be better if the navigation menu was always at the left of the window and each site appeared in a frame at the right. Create a new HTML file that sets up the web page using Frames. There is an example of this in the `ImageMap Solution` folder called `Applet-Frames.html`.

Double Buffering

Rather than draw directly to screen, the applet code uses a technique known as double buffering to minimize the flickering. The applet draws to an off-screen location (in memory) and then, when

the image is fully drawn, draws the whole image on the screen at once.

Change the code so that it does not use double buffering to see what happens. To make this change, remove the code indicated by the comments in Listing 3.3.

Listing 3.3 Screen flicker

```
public class ImageMapApplet extends Applet
{
  private Image image, hImage // remove this ->, offscreenImage;
//...

public void init()
  {
//....

// remove the following line
offscreenImage=createImage(imgWidth, imgHeight);
```

New parameters

The way the applet is currently set up, you need to recompile it every time you change the graphic, whether you add items, delete items, use different sizes, etc. The code that sets this up is `initImages()`. It would be nice if you didn't have to do this. It would make the applet more usable and portable to just change the parameters in the HTML file and leave the source code alone (Listing 3.4).

Listing 3.4 HTML area parameters

```
<PARAM name="sections" value="4"> // the number of highlighted
sections in the image
<PARAM name="coords0" value="0, 0, 428, 60">// the coordinates for
each
<PARAM name="coords1" value="0, 60, 428, 60">
<PARAM name="coords2" value="0, 120, 428, 60">
<PARAM name="coords3" value="0, 180, 428, 60">
```

Then you would need a method to parse these coordinates (Listing 3.5).

Listing 3.5 Value parser shell code

```
String temp;
temp=getParameter("sections");
int s=Integer.parseInt(temp);
for (int i=0; i<s; i++){
  temp=getParameter("coords"+i);
  // parse string for numbers here
  // this can be done using methods from the String and Integer
classes
  Do_Clip_Image(x, y, w, h);
}
```

4

Creating Java Projects

This chapter explains how to create and manage any Java project, and how to set the options in the Java Project settings panels.

This chapter contains these sections:

- Types of Java Projects describes the types of Java projects you can create with CodeWarrior.
- Using Project Stationery describes each project stationery option in the New Projects dialog box.
- Working with Java in CodeWarrior describes how to create and manage any Java project from start to finish.
- Using the classes.zip Library describes what's in classes.zip, the library file that's in most every Java project, and how to add it in a project.

Types of Java Projects

This section discusses the various types of code objects that you can create for Java. They include the following:

- Applets
- Applications
- Libraries

Applets

An Applet is a small Java application that runs over the Web, usually through a Web browser. Applets can consist of multiple class files or a single Jar file.

Applications

A Java Application are executables that usually run through a Java Interpreter. Applications can consist of multiple class files or a single Jar file.

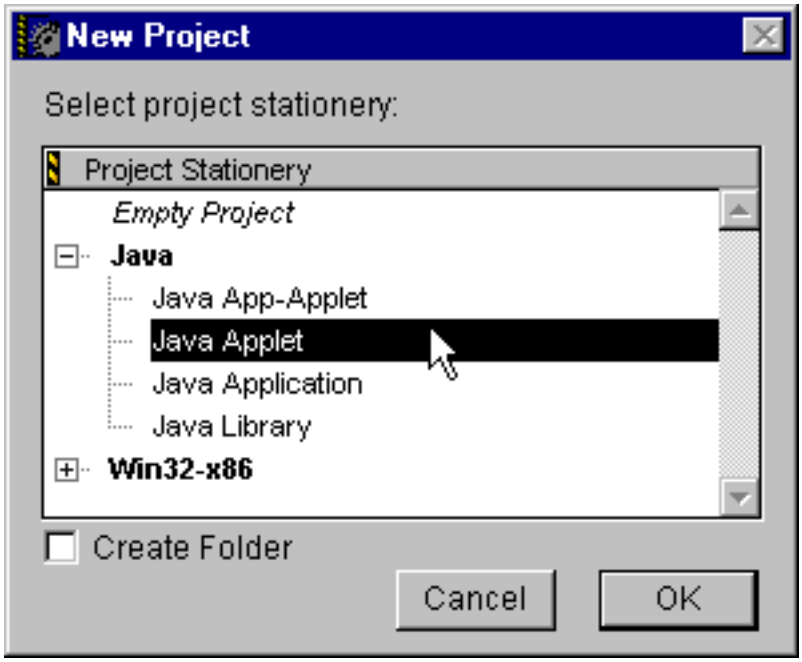
Libraries

A Java Library is a collection of Java classes that can be used in many projects. Libraries are usually compiled into zip files (like `classes.zip`) or Jar files.

Using Project Stationery

The **New Project** dialog box (Figure 4.1) displays a list of several different pre-configured projects and their associated files, called *Project Stationery*.

Figure 4.1 New Project dialog box



Each set of project stationery is pre-configured for different types of Java projects. This saves you time when creating your own project files as you only need to change a few options instead of setting up a project from scratch.

Table Table 4.1 lists each project stationery option.

Table 4.1 Project stationery options

Stationery	Description
Java Applet	Pre-configured settings for a Java applet. Output is a Jar file.
Java Application	Pre-configured settings for a Java application. Output is a Jar file.
Java App-Applet	A multi-target project (Applet & Application) that demonstrates how to use `AppletFrame.java` to turn your applet into an application.
Java Library	Pre-configured settings for a Java library. Output is a Jar file.

Working with Java in CodeWarrior

This section describes how to create, run, and debug any kind of Java project, including applets, applications, and libraries. The topics in this section are:

- Creating a New Java Project
- Creating Java Code
- Changing Settings
- Running a Java Project
- Debugging a Java Project
- Kinds of Application Projects

Creating a New Java Project

The following procedure describes how to create a new Java project. The project can be either an applet, an application, or a library of Java code.

1. **Create a new project.**

   Choose **New Project** from the File menu. CodeWarrior displays the New Project dialog.

2. **Select project stationery.**

   In the New Project dialog box, select the proper stationery for the kind of project you want to create: applet, application, or library. To learn more about various kinds of application projects, see "Kinds of Application Projects" on page 64.

   You may also decide whether to have CodeWarrior create a new folder for you in which to place all your files. Typically you want to create a new folder.

   When you click the **OK** button, CodeWarrior displays a file dialog.

3. **Name the project.**

   In the file naming dialog, enter a name for the project, and click **OK**.

   If you use a suffix, it must be `.mcp`. If you don't use a suffix, CodeWarrior adds it for you.

TIP: If you want to compile your project with all versions of CodeWarrior, the project suffix *must* be `.mcp`.

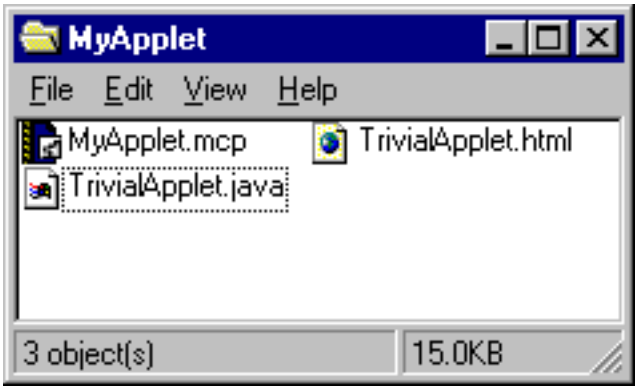
If you had the **Create Folder** check box on in the New Project dialog box, CodeWarrior creates a new folder. Inside that folder, CodeWarrior puts a new project file (with all project settings identical to the stationery project you selected), and duplicates any files included in that stationery project.

For example, for an applet, you get a new project file, an HTML file, named `TrivialApplet.html`, and a Java source file named

`TrivialApplet.java`. Figure 4.2 shows you the result when you use the Java applet stationery to create a project named MyApplet.

The project file has the name you entered in the dialog box. The new folder has the same name as the project file without any naming extension.

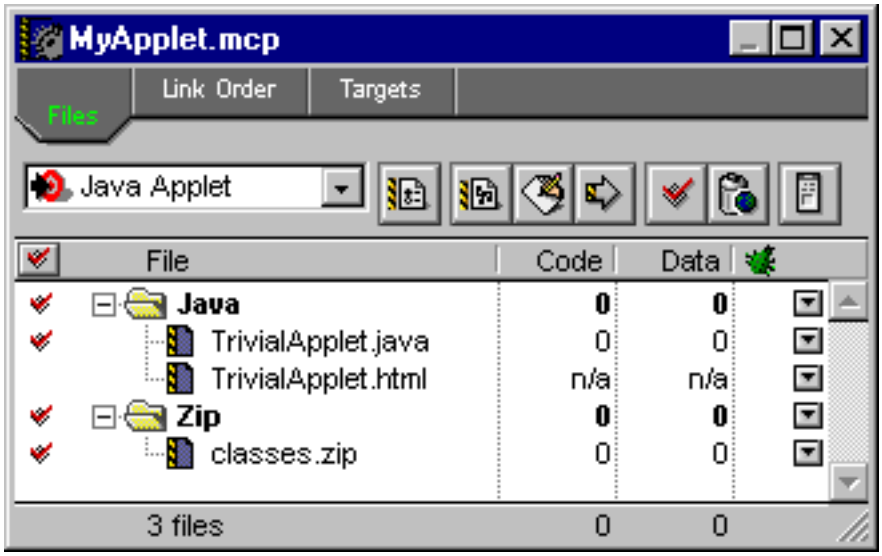
Figure 4.2 The Java Applet folder



WARNING! The project manager also creates a directory named MyProject Data, where "MyProject" is the name of your project. CodeWarrior uses this directory to store internal data. You should not modify or move any file in the data directory.

After creating the files and folder, CodeWarrior displays a new project window, like the one below:

Figure 4.3 The Java Applet project



The stationery projects hold example files. In typical use you modify these files and save them with a new name, or replace them with your own files.

The `classes.zip` file is a shared library of standard Java classes your applet can use. For more information, see "Using the classes.zip Library" on page 64.

Creating Java Code

To create Java code, use the CodeWarrior tools: the editor, the browser, and the project manager. For details on these tools, see the *IDE User Guide*.

In typical practice, you start with a project based on stationery, which includes example or "placeholder" files. To learn how to create a project in this way, see "Creating a New Java Project" on page 58.

For example, if you are creating an applet based on the Java applet stationery, there are two such files: `TrivialApplet.java` and `TrivialApplet.html`.

The `TrivialApplet.java` file contains a sample Java applet. You can open the file, replace its contents with your own code, and save the file under a new name. You can add any other source file as well.

NOTE: Each Java file must contain only one public class. The file name must be the same name as the public class with the suffix `.java`. For example, if the name of your class is MyClass, the Java file must be named `MyClass.java`. If you don't follow this convention, CodeWarrior may compile your project anyway, but other Java compilers may not.

Similarly, the `TrivialApplet.html` file contains sample HTML code. You can modify this file so that it displays your applet. You typically specify both the location of the applet (a folder name) and the name of the class containing the main function. You can set the name of your classes folder by opening the **Target Settings** dialog box, and modifying the name specified in the **Output** panel. See "Target Settings" on page 74 and "Java Output" on page 86 for more information.

Changing Settings

If you use project stationery as a starting point, you generally won't need to change many of the settings. The default settings are perfect for most projects. However, this is not the case for all settings or all projects. This is especially true if you have more than one target in a project. You can modify a variety of settings for Java projects. Each panel is explained in detail in "Target Settings for Java" on page 73.

Common settings you might change are the main class (for Java applications) and the output name (the name of your final program). For applications you may also choose whether to output your files in a class folder, or in a zip file.

Specifying the main class

To tell CodeWarrior the name of your main class for a Java application, go to the **Java Target** panel, make sure that you are creating an application, and set the name in the **Main Class** field in the panel. For more information, see "Main Class" on page 78.

Specifying output name

To tell CodeWarrior what to name your final output file (be it an applet or an application), go to the **Java Output** panel and enter the name in the **Name** field. For more information, see "Java Output" on page 86.

Specifying output type

To tell CodeWarrior whether to create a class folder or a Jar file, go to the **Java Target** panel and set the Type option. For more information, see "Java Target" on page 76.

Running a Java Project

You may run applets and applications. You cannot run libraries. To run a project, choose **Run** from the Project menu. At that time, CodeWarrior compiles and links any changed files, and launches your program. Precisely how your program launches depends upon whether it is an applet or an application.

Running a Java applet

When developing an applet, you typically have a small HTML file that embeds the applet code. CodeWarrior uses this file to run your program from within the IDE. When you choose **Run** from the Project menu, CodeWarrior opens the first HTML file in the Link Order view with an HTML browser or applet viewer. In other words, the order of HTML files in the File view in the Project Window does not matter. It is the order as shown in the Link Order view in the Project Window.

The default browser is Internet Explorer. To learn how to modify this setting, see "Applet" on page 77.

NOTE: For applet projects, if there is no HTML file in your project, the **Run** command is disabled.

You can add as many HTML files as you want to a project. If your applet is complex, you may want several HTML files that test different aspects of your applet. To change the HTML file that CodeWarrior uses, simply rearrange the files in the Link Order view of your project window.

NOTE: You can also run an applet by opening one of its HTML files with any Java-enabled browser.

Running a Java application

To run a Java application, choose **Run** from the Project menu in the CodeWarrior IDE. The IDE launches `jview`, passing it the class name specified in the **Java Target** panel. For information on how to specify the main class, see "Main Class" on page 78.

Debugging a Java Project

You debug an applet project the same way you debug any other project. For more information on the CodeWarrior debugger, consult the *Debugger User Guide*. For more information on Java-specific features in the debugger, see "Debugging Java Projects" on page 93.

To debug a Java project, follow these steps:

1. **Turn on debugging.**

   This step can be skipped if debugging is already enabled.

   Choose **Enable Debugger** from the Project menu. You may see a dialog box telling you that certain settings must be modified for debugging. Click OK, and CodeWarrior sets up debugging automatically. In the project window, black dots appear in the debugging column next to the source files.

2. **Run the project under the debugger.**

   Choose **Debug** from the Project menu. CodeWarrior compiles any changed files and generates debugging information for all files that have a black dot in the debugging column of the project window. CodeWarrior launches your project using the appropriate mechanism for your project. For a discussion of this process, see "Running a Java Project" on page 62.

   Once the project is running, control returns to the debugger. You can now debug the applet like you would any other program. For more information on the debugger, see the *Debugger User Guide*. For more information on Java-specific features in the debugger, see "Debugging Java Projects" on page 93.

Kinds of Application Projects

CodeWarrior lets you create Java applications that you can run with the `jview` utility included with Microsoft Windows 95 and Windows NT 4.0. CodeWarrior can generate these types of output files:

- A *jar file* is an archive of all your classes. It is a convenient way to transfer and organize all the class files and any associated other files in one package. It can be compressed or uncompressed.
- A *class folder* is a folder that contains all your classes. Choose it if you need to run the file on other computer platforms that can't use jar files.

For more information on setting the kind of output files you can use for an application, see "Specifying output type" on page 62.

All these files types contain Java bytecodes that will run on any computer platform that supports Java.

Using the classes.zip Library

The `classes.zip` file is a library of Java classes that all Java programs share. It includes all the packages in the Java API, such as the Abstract Windowing Toolkit (AWT), I/O, and Applet packages.

It's located inside the `Java` folder of the System folder (either `Winnt` or `Windows`).

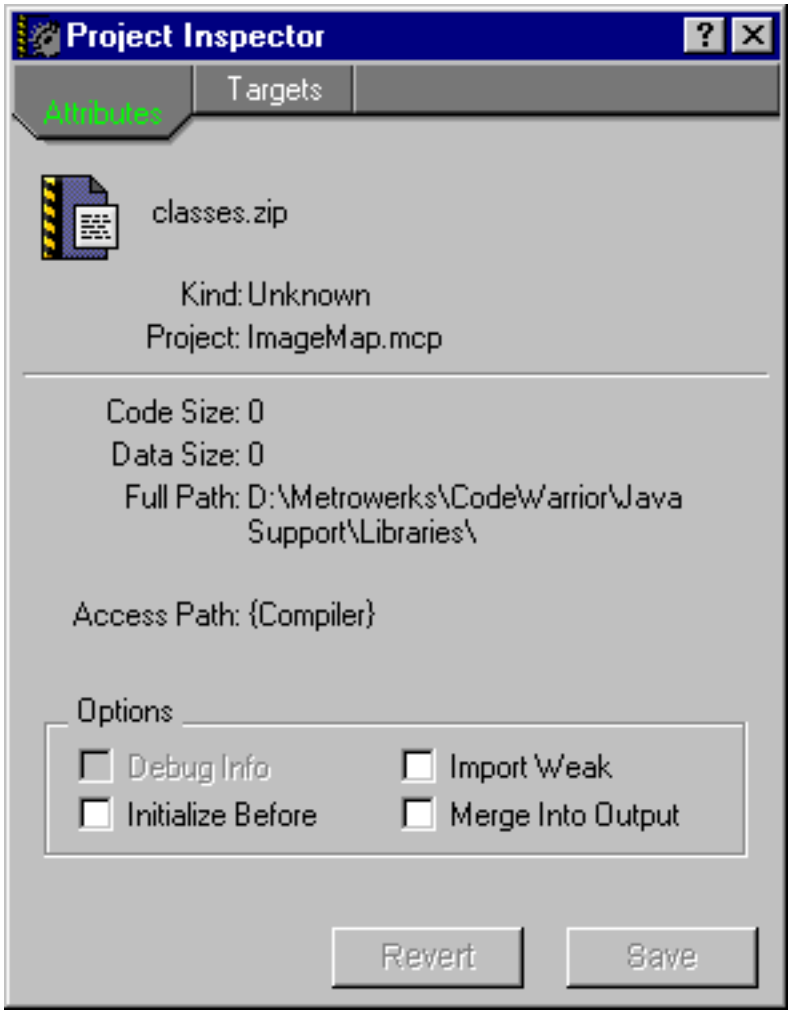
Almost every Java program that you create uses `classes.zip`. However, `classes.zip` is a large file, about 1.5 MB, so you probably don't want to copy it into every one of your Java applications and applets.

To instruct CodeWarrior not to merge the `Classes.zip` file into the final output, first select the `Classes.zip` file in the project window. Then choose **Project Inspector** from the Window menu to open the **Project Inspector**. Make sure that the **Merge Into Output** checkbox is not selected. CodeWarrior won't copy the library into your finished program.

To enable your project to compile without the `Classes.zip` library, make sure that the **Import Weak** checkbox is selected. With these settings, your program will automatically look for the library on the host system when it is executed.

The **Project Inspector** window should now appear as in Figure 4.4.

Figure 4.4 The Import Weak Classes Setting



When you create a project with project stationery, `classes.zip` is automatically set up this way for you.

NOTE: The **Initialize Before** option is ignored by the CodeWarrior Java compiler. **Merge Into Output** will add `Classes.zip` to the final binary.

5

JavaDoc

JavaDoc is a batch compiler written in Java by JavaSoft™. It processes Java source code files, and uses the comments preceding classes, methods, etc. to generate HTML based documentation for the code. The Java API documentation on your CodeWarrior CD was generated by this compiler.

There is one primary topic discussed in this section:

- CodeWarrior JavaDoc Implementation
- Using JavaDoc

CodeWarrior JavaDoc Implementation

CodeWarrior implements JavaDoc as a Pre-Linker because of the batch nature of the JavaSoft compiler. The JavaSoft version of JavaDoc must know about all the compiled files simultaneously. In the current CodeWarrior plugin architecture, a compiler works on one file at a time, but a linker knows about all compiled files. So JavaDoc in CodeWarrior is implemented as a pre-linker.

Another added benefit of implementing JavaDoc as a pre-linker is speed. Creating JavaDoc files should be the last thing you do in with your Java project. Assuming this is true, your code will have already been compiled and tested. At this point, when you implement JavaDoc, the process is very fast as everything the pre-linker needs to know was already done at compile time. No need to recompile, just relink.

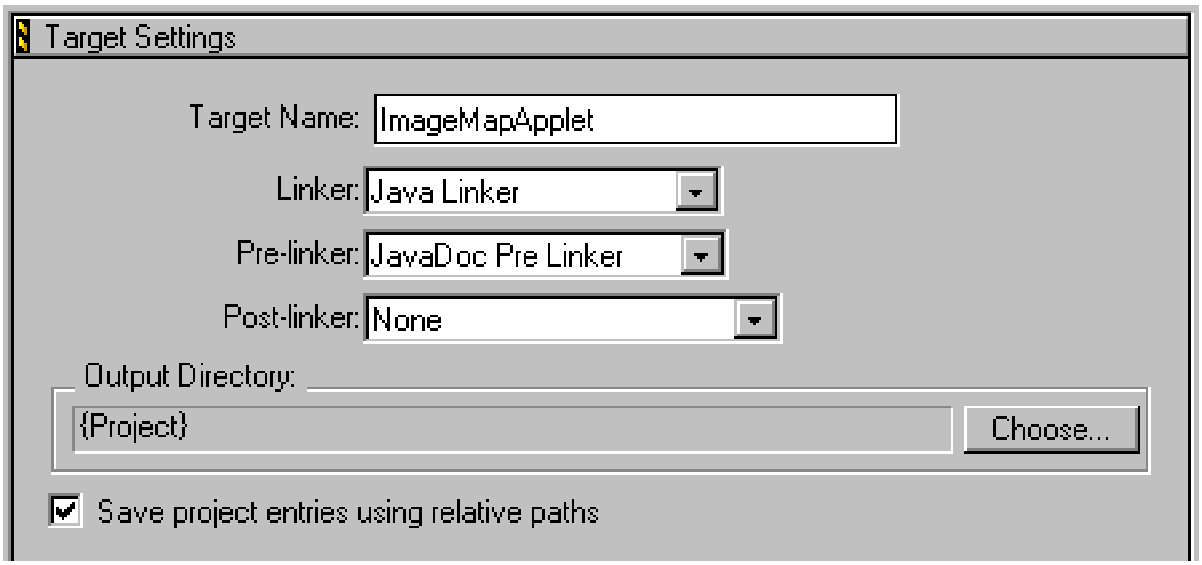
Using JavaDoc

Using JavaDoc is as simple as choosing the JavaDoc pre-linker, and compiling the files.

1. **Choose pre-linker for JavaDoc target.**

   Choose ***Target* Settings** from the Edit menu, where ***Target*** is the name of your current Java target. Choose **JavaDoc Pre Linker** from the **Pre-Linker** popup menu in the Target Settings panel (Figure 5.1). Click Save to save changes. Close the Target Settings dialog box. Click **OK** for subsequent warning dialogs.

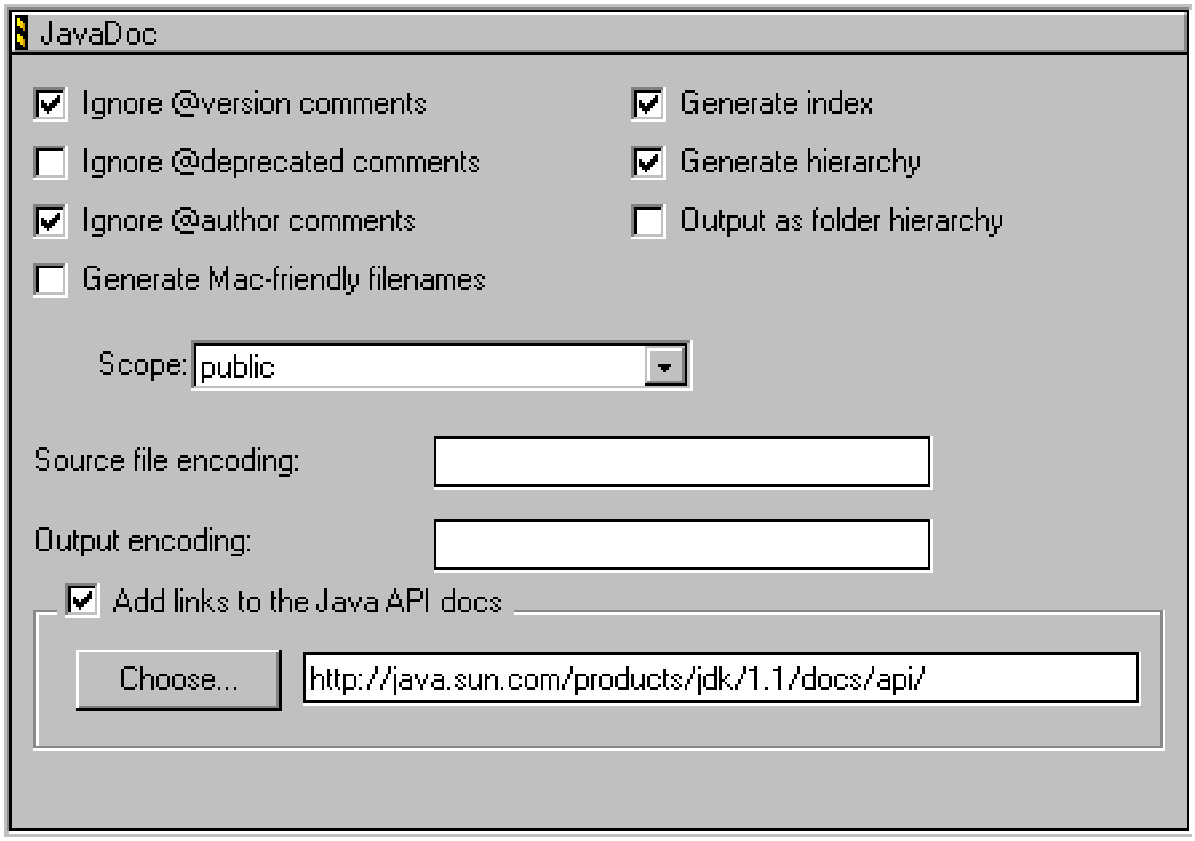
Figure 5.1 Choosing JavaDoc linker



2. **Set JavaDoc Project options.**

   Choose the settings you wish to use for JavaDoc from the **JavaDoc** panel (Figure 5.2). For a full description of each setting in this panel, see "JavaDoc" on page 89.

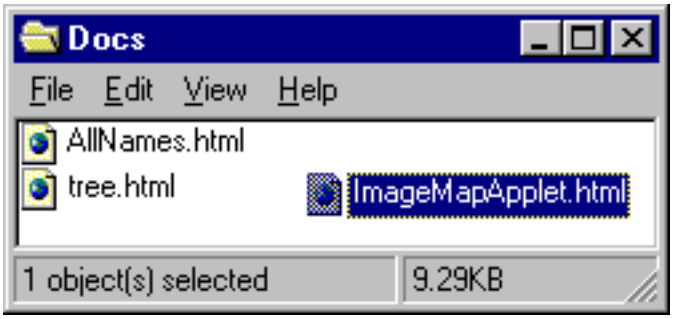
Figure 5.2 JavaDoc Project panel



3. **Compile the project.**

   Choose **Make** from the Project menu. JavaDoc creates a new folder in the same folder as your project called "Docs" (Figure 5.3).

Figure 5.3 JavaDoc folder

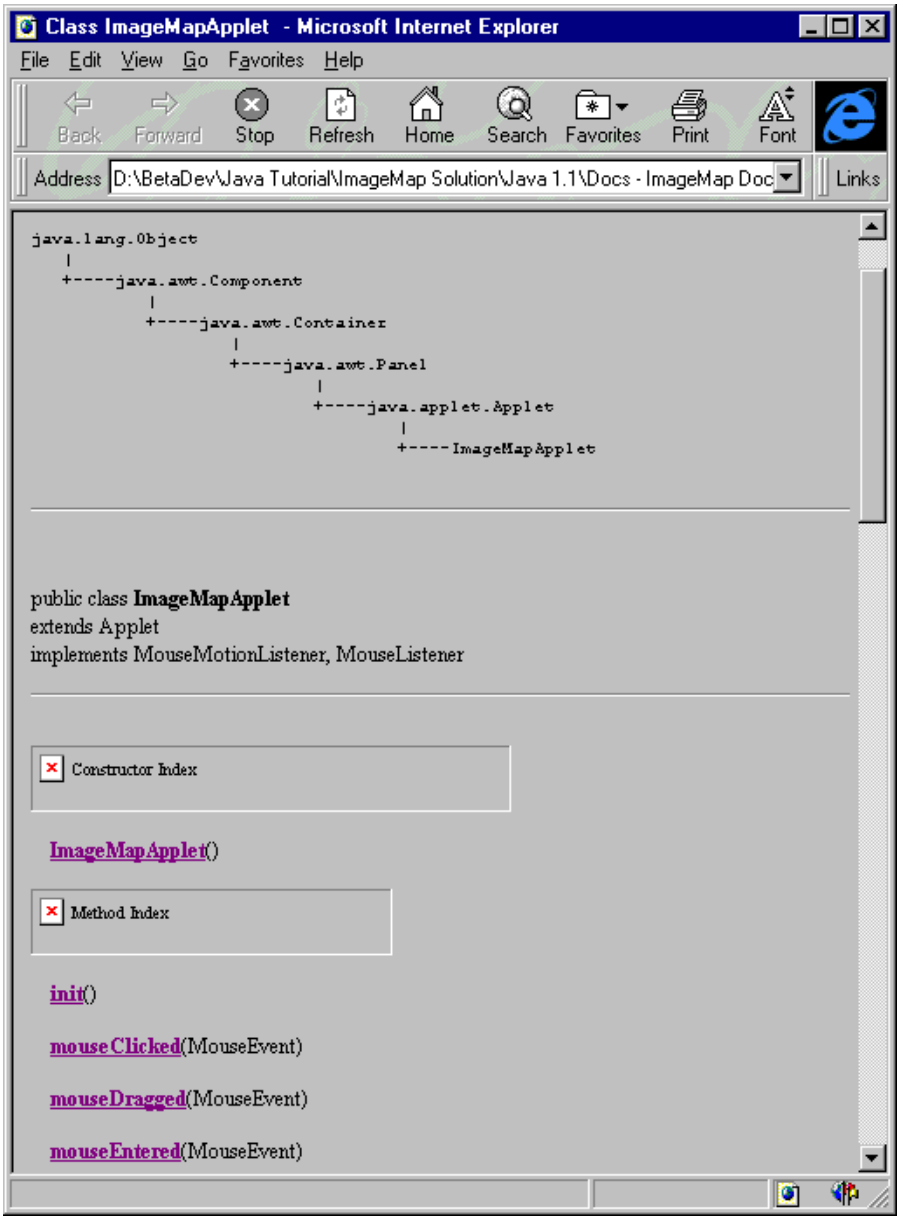


The last step is to open the HTML files in your preferred HTML Browser (Figure 5.4). A separate HTML file is generated for every public Java class included in the target. There are also some default files generated depending on the structure of your Java project.

You may find some links do not work. This is because your project does not include this information. For example, if your project doesn't include packages, the link to `Packages.html` fails because the file was not generated by JavaDoc.

TIP: Images are not automatically generated by JavaDoc, though the HTML source that references these images is generated. You may create your own images to use, use the images from the Java API documentation, or you can use the images supplied with your CodeWarrior installation. The images are located in: `Metrowerks CodeWarrior\Java Support\Metrowerks\JavaDoc\images.`

Figure 5.4 ImageMap doc class



6

Target Settings for Java

CodeWarrior *target settings* are options you can specify to determine various aspects of your project's behavior, such as how it compiles and links. Choosing the right settings can significantly improve the size and speed of your final code.

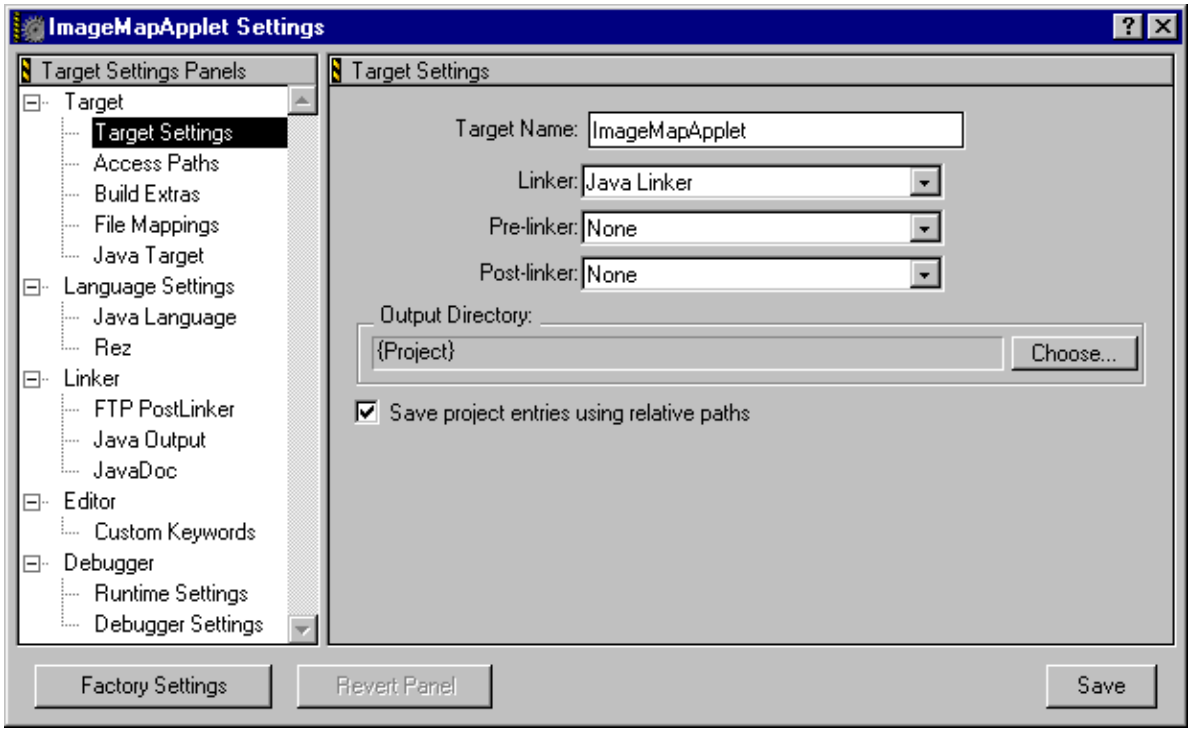
Target settings are organized into *panels* that you can display in CodeWarrior's project settings window. Different settings panels control various properties of the project. The Target Settings dialog box is shown in Figure 6.1.

This chapter discusses only those project settings panels that relate specifically to Java programming:

- Target Settings
- Java Target
- Java Command Line
- Java Language
- FTP Post Linker
- Java Output
- JavaDoc

See the *IDE User Guide*, *C Compilers Reference*, and *Pascal Compilers Reference* for information about other settings panels available in the Target Settings Dialog.

Figure 6.1 Target Settings Dialog



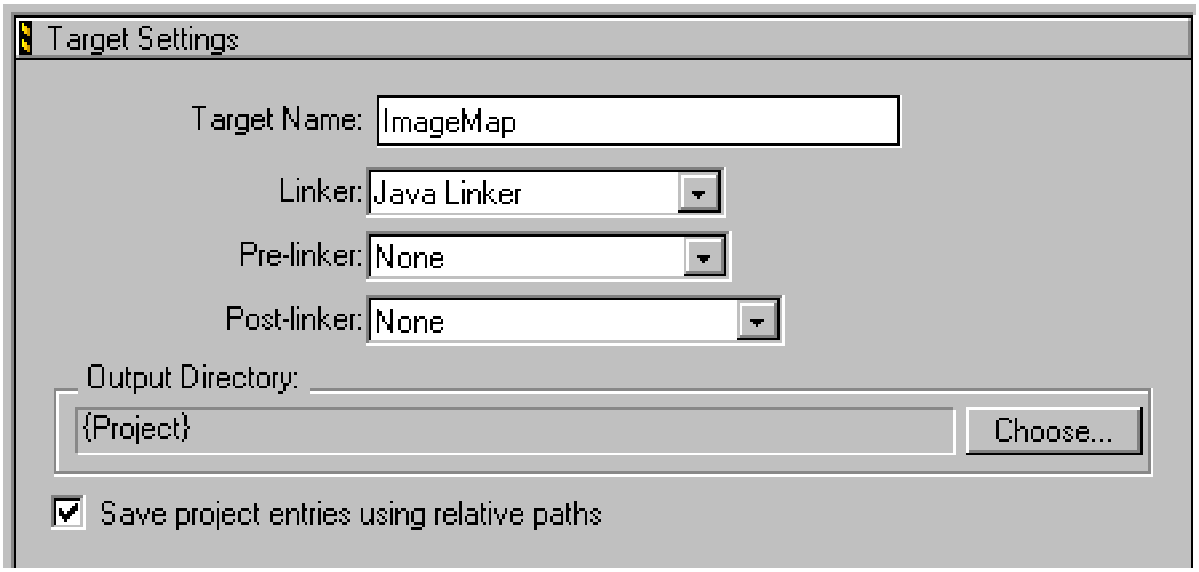
Target Settings

The **Target Settings** panel is the most critical panel in CodeWarrior. This is the panel where you pick your target operating system and/or microprocessor.

The **Target Settings** panel, shown in Figure 6.2, allows you to set the name of your target, as well as which linker and post linker plugins to use for the target. When you select a linker, you are specifying the target operating system and/or chip. The other panels available in the Settings dialog box will change to reflect your choice.

Because the linker choice affects the visibility of other related panels, you must set your target first before you can specify other target-specific options like compiler and linker settings.

Figure 6.2 Target Settings Panel



The items in this panel are:

Target Name	Post-linker
Linker	Output Directory
Pre-linker	Save project entries using relative paths

Target Name

Use the **Target Name** field to set or change the name of a target. When you use the Targets view in the Project window, you will be able to see the name that you have set.

This is not the name of your final output file. It is the name you assign to the target for your project file to use. The name of the final output file is set in the Java Output panel.

Linker

Choose a linker from the items listed in the Linker pop-up menu. There is one Java linker.

Pre-linker

Some targets have pre-linkers that perform additional work (such as a data format conversion) before linking. There is one pre-linker for Java: the JavaDoc Pre-Linker.

See also "JavaDoc" on page 67.

Post-linker

Some targets have post linkers that perform additional work (such as a data format conversion) on the final executable. Based on this setting, corresponding panels will be added to the **Linker** section of the panel list on the left-hand side of the Target Settings dialog. For instance, if the **JCommand Line** linker is selected, a **Java Command Line** post linker panel will appear under the **Linker** section.

The post linkers available for your use are described in detail on the following pages:

FTP Post Linker "FTP Post Linker" on page 84

JCommand Line "Java Command Line" on page 79

Output Directory

This is the directory where your final linked output file will be placed. The default location is the directory that contains your project file. Click the **Choose** button to specify another directory.

Save project entries using relative paths

Set this option if you have Java files that use the same name, but are physically located in a different directory. This setting allows the IDE to recognize it as a different file. If you do not have files with the same name, you can leave this option off.

Java Target

The settings you can specify on the **Java Target** panel depend on whether you're creating an Applet, a library, or Jar file.

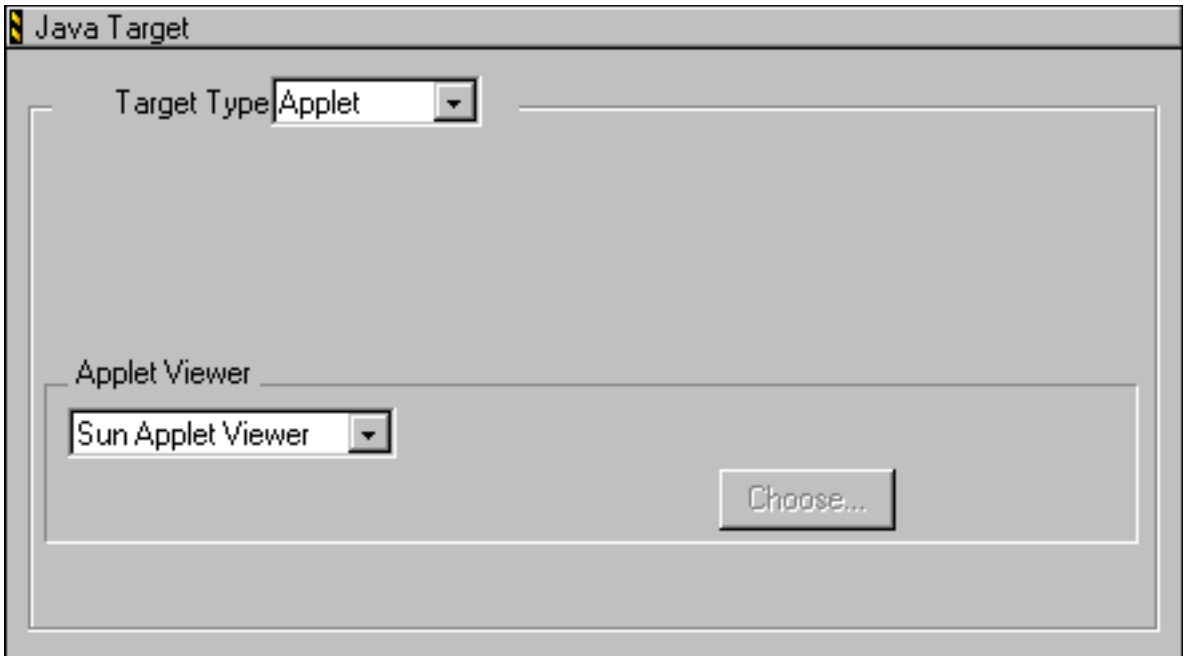
This section describes settings available for each project type:

- Applet
- Application
- Library

Applet

If the project **Type** is **Applet** (Figure 6.3), you have the choice of using a default applet viewer or choosing your own. Click the **Set** button to the right of the **Applet Viewer** edit field to choose a different applet viewer.

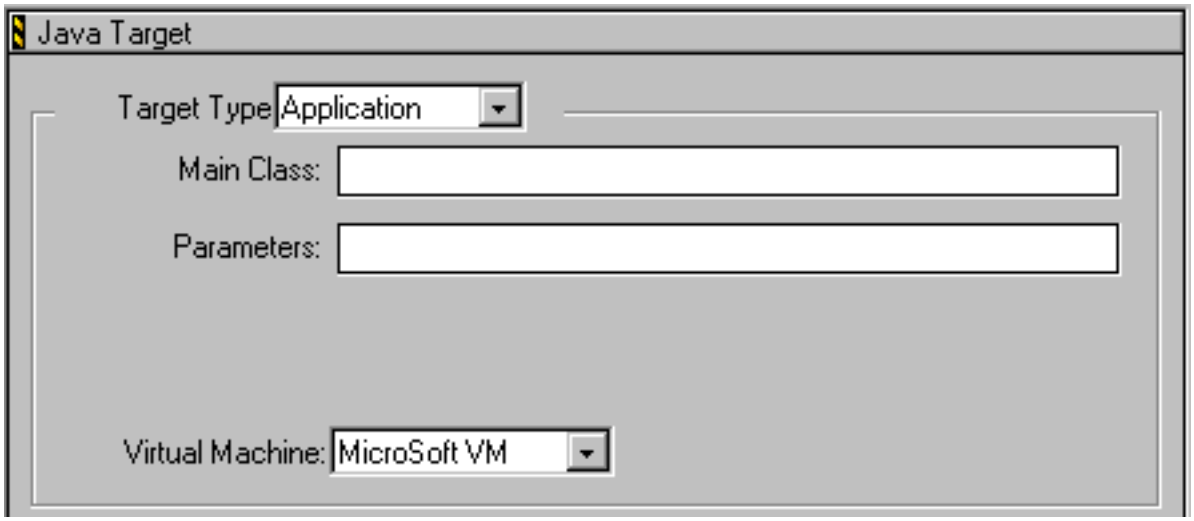
Figure 6.3 Applet project type



Application

If the project **Type** is **Application** (Figure 6.4), you need to specify your main class and any parameters your class requires.

Figure 6.4 Application project type



Main Class

The **Main Class** field identifies the name of the class containing the `main()` method in a Java application. Enter only the name of the class, do not use the `.class` extension. You are specifying the class itself, not the object file.

CodeWarrior passes this name to the VM running the application so it can begin execution. This field must have a value in order to run a Java application.

NOTE: the `main()` method must be declared `public static void main(String args[])`, and must be in a public class. Also, the capitalization of the name in the **Main Class** field must match the capitalization in the Java file. In Java, `myclass` and `MyClass` are *different* classes.

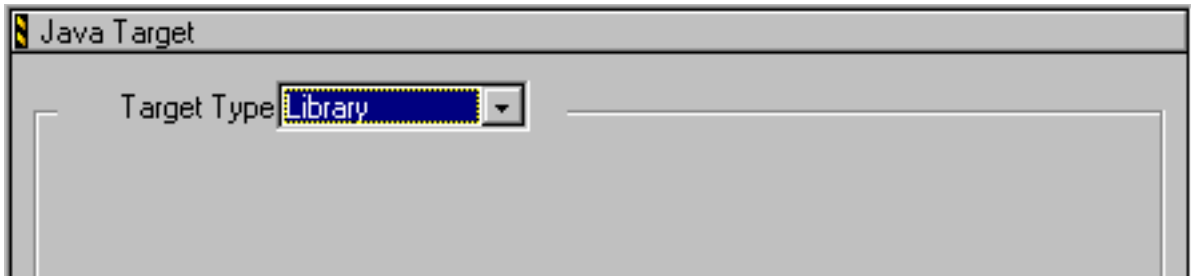
Parameters

Enter any arguments you wish to pass to the main() method when the application launches. A series of arguments must be delimited by spaces.

Library

A project Type of **Library** (Figure 6.5), does not have any further options to set.

Figure 6.5 Library project type



Java Command Line

The Java Command Line panel allows you to launch a Java application after your project is successfully built to do further processing of your project output.

For instance, you may use it to launch RMIC, a post linker that takes .class files and generates stub classes. The java linker would generate the .class files. Then the RMI compiler (RMIC) would take those class files and generate stub class files for use with RMI.

The panel contains the following items:

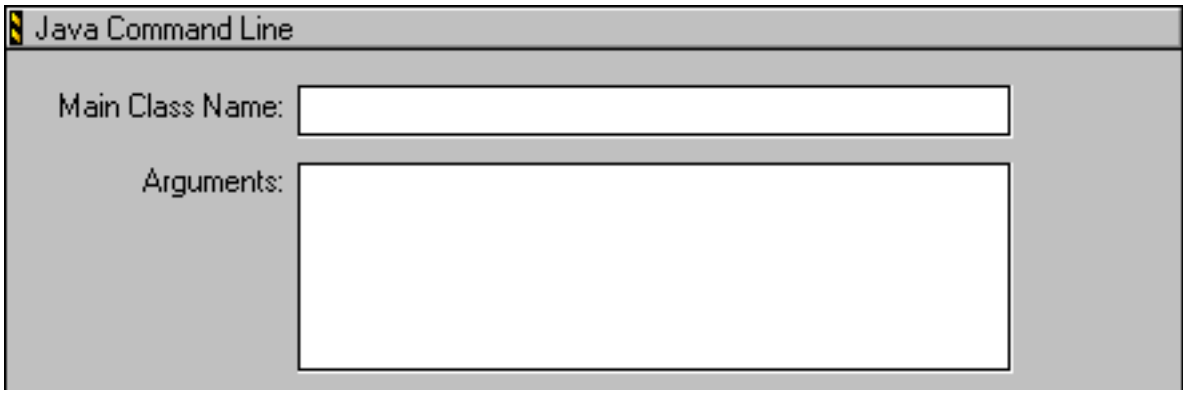
Main Class Name

Put the name of the main class of the application you want to be invoked into this field. Once your project is successfully built, the linker will run the application.

Arguments

Put any arguments to be sent to your application into this field.

Figure 6.6 Java Command Line panel



Currently, this linker adds paths to all zips and jars in the project to the VM classpath. It also adds the project output to the classpath.

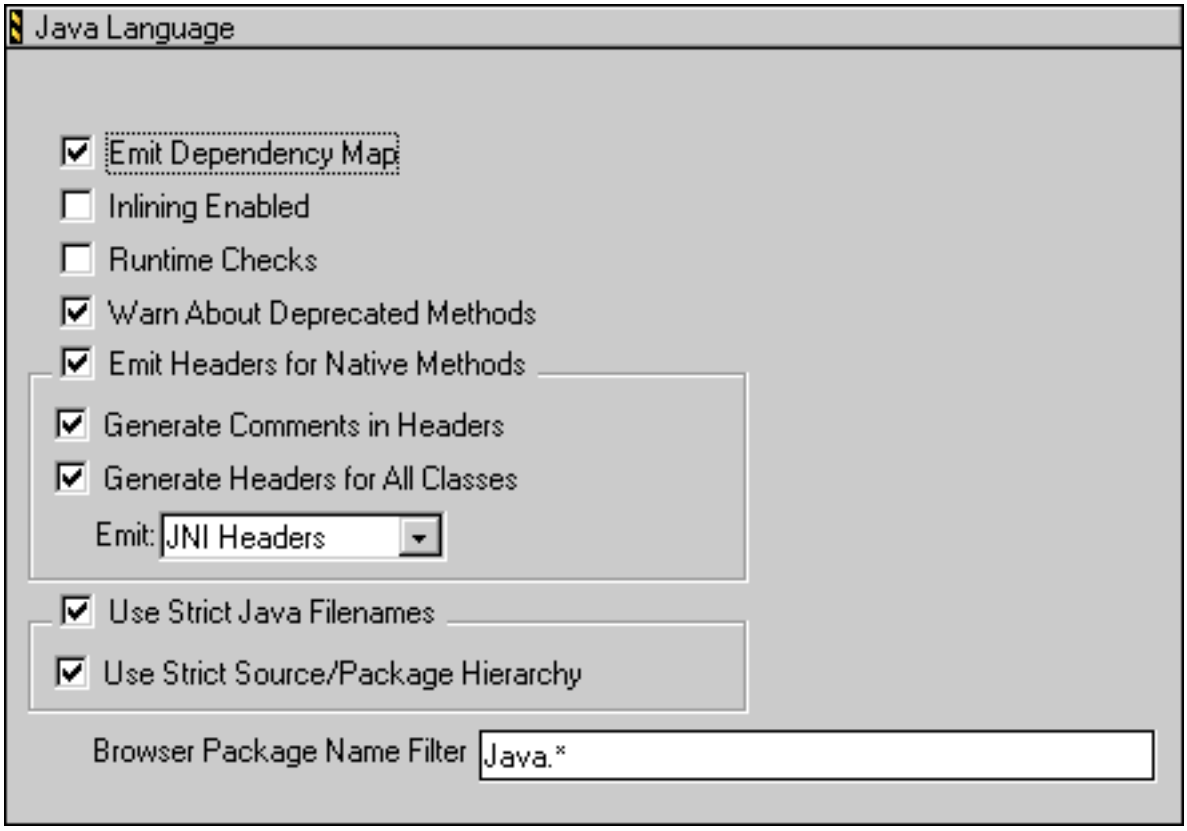
NOTE: Although this linker can execute any Java application specified in the command line settings, its intended use is to invoke java command line post linker type tools (such as RMIC, obfuscators, etc.). Therefore, there is no support for AWT based apps nor apps which make use of System.in.

WARNING! Since it is impossible for the post linker to know what type of application it will be invoking, it is the responsibility of the user to make sure that only the appropriate type of command line tools are run using it.

Java Language

The **Java Language** panel (Figure 6.7) contains project settings related to code generation for the Java platform.

Figure 6.7 Java Language panel



These options in this panel are:

Generate Profiler Information	Emit Headers for Native Methods
Emit Dependency Map	Use Strict Java Filenames
Inlining Enabled	Use Strict Source/Package Hierarchy
Runtime Checks	Browser Package Name Filter
Warn About Deprecated Methods	

Generate Profiler Information

To use this option, you must include `Profiler.zip` in your project. This option adds timing information to your Java methods. When your Java code finishes executing, a new file is created with the timing data. You need to copy this file to a Mac OS computer. You can then view this file with the MW Profiler application.

NOTE: You cannot profile Applets. Only Applications.

Emit Dependency Map

This option is similar to generating a link map in other languages such as C and C++. When enabled, a new file is created in the same directory as your project file called *`sourcefile`*`.JMAP`, where *`sourcefile`* is the name of your `.java` source file. The JMAP file lists all the class dependencies for each class in the file.

For example, the statement `import java.*` is common in Java source code. The dependency map tells you the *exact* class dependencies. So the statement `import java.*` may become:

```
import java.applet.Applet;
import java.applet.AppletContext;
```

You can then use these statements in your Java source files instead of importing everything. This makes things cleaner, faster and more portable to other Java compilers.

Inlining Enabled

Allows inlining of smaller methods where appropriate.

Runtime Checks

Please read the release notes for information on this option.

Warn About Deprecated Methods

Gives warning messages if you are using any depreciated methods.

Emit Headers for Native Methods

Emits headers for any native methods for *every* class in your project. When this option is enabled, the **Emit** popup menu appears allowing you to choose between **JNI Headers** and **Sun VM Headers**. JNI is the newest and preferred method. However, for compatibility reasons, you may wish to choose the **Sun VM Headers** option.

The other options that can be used in conjunction with Emit Headers for Native Methods are **Generate Comments in Headers** and **Generate Headers for All Classes**.

Generate Comments in Headers moves comments from the java sources files to the native headers.

Generate Headers for All Classes generates the headers in a tree (like sun/tools/java/Identifier.h) instead of mangled with the package (like sun_tools_java_Identifier.h)

Use Strict Java Filenames

Forces strict class names based on the Java source file names. For example, if you have a java source file called FooBar.Java, the compiler expects to find a class called FooBar.class in this file. No other class can be in this file.

NOTE: This option should be used in conjunction with Use Strict Source/Package Hierarchy.

Use Strict Source/Package Hierarchy

Forces strict adherence to source and package hierarchy paths. For example, if you have the following package:

```
Foo\bar\seam
```

The compiler expects a to find file `seam.java` in a directory called "`bar`" which is in a directory called "`Foo`."

This option is only available when Use Strict Java Filenames is enabled.

Browser Package Name Filter

Accepts a semicolon delimited list of names that you do not want to show up in the class browser. If you type in: `java.io; java.lang`, the browser will only display `File`, `Object` and `reflect.Method` for classes like `java.io.File`, `java.lang.Object`, and `java.lang.reflect.Method`.

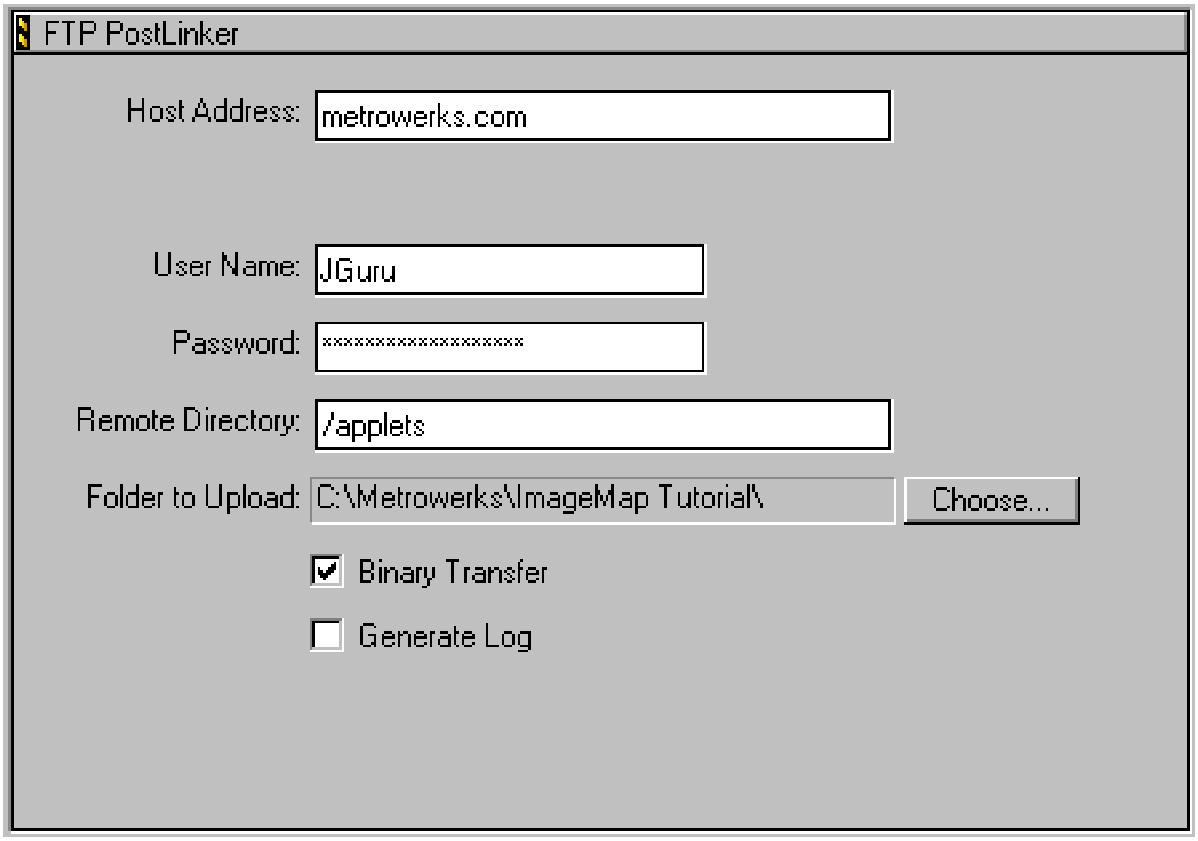
You can also use wildcards like `java.*`, in which case the browser will strip off the entire package for any class starting with "java."

Finally, you can also type in "`*`" (no quotes), in which case the browser strips all packages off everything.

FTP Post Linker

The **FTP Post Linker** panel allows you to move a folder containing your completed Java binary and any associated files to a server to be accessed through the World Wide Web.

Figure 6.8 FTP Post Linker panel



The items in this panel are:

Host Address	Folder to Upload
User Name	Binary Transfer
Password	Generate Log
Remote Directory	

Host Address

The host address of the server you want to upload your Java files to.

User Name

Your user ID on the host server.

Password

Your password on the host server.

Remote Directory

Directory you want to upload the Java files to on the host system. You must have access privileges for this directory.

Folder to Upload

The full path to the folder on your local hard drive containing the files to upload. There is no way to specify that individual files be uploaded, so the linker will transfer the entire contents of the folder specified here. Make sure you do not have any files that you do not want transferred in this folder.

Binary Transfer

Transfer the files to the host system using Binary mode instead of ASCII. It is recommended this option always be turned on to avoid any transfer problems.

Generate Log

Generates a text log file of the transfer. Any errors or problems are recorded in this file.

endrange marker for the FTP PostLinker panel.

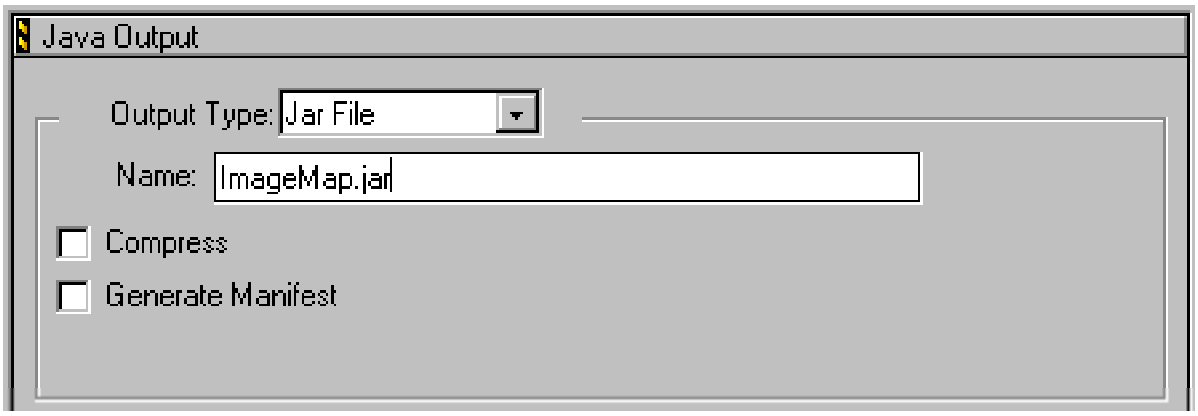
Java Output

The settings on the **Java Output** panel (Figure 6.9) control the operation of the CodeWarrior linker for Java. The **Output Type** can be:

- Class Folder
- Jar File

This section describes the settings for each option in detail.

Figure 6.9 Java Output panel

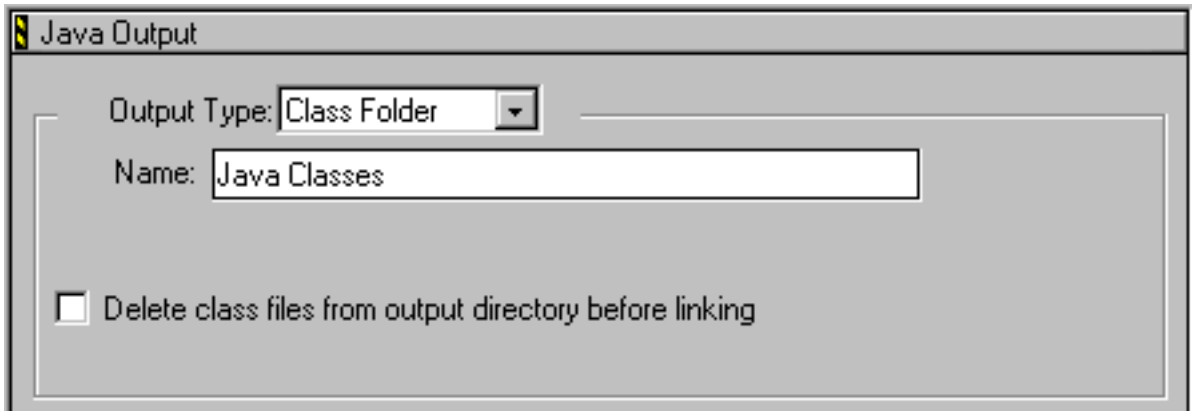


Class Folder

If you choose **Class Folder** from the **Output Type** popup menu (), the IDE will create a new folder with the name you specify in the **Name** field. Every class in your project will be placed in this folder.

Delete class files from output directory before linking simply removes all class files in the target directory before linking your code. This is useful if you recompile your source often as it makes sure no old code exists in the target folder.

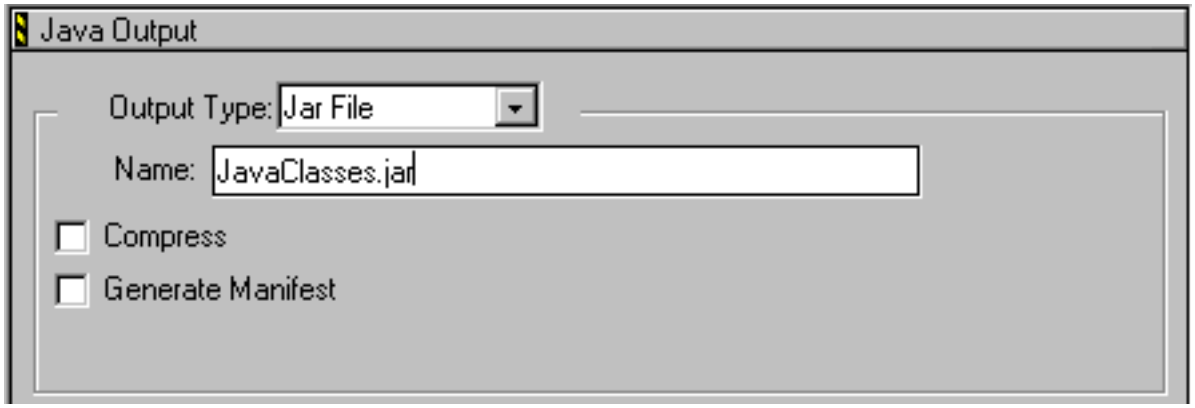
Figure 6.10 Output as class folder



Jar File

A Jar file is a Java Archive file. The **Jar File** option (Figure 6.11) allows you to control how the Jar file is created. Choose **Compress** to create a compressed Jar file. Choose **Generate Manifest** to create manifest information in the Jar file.

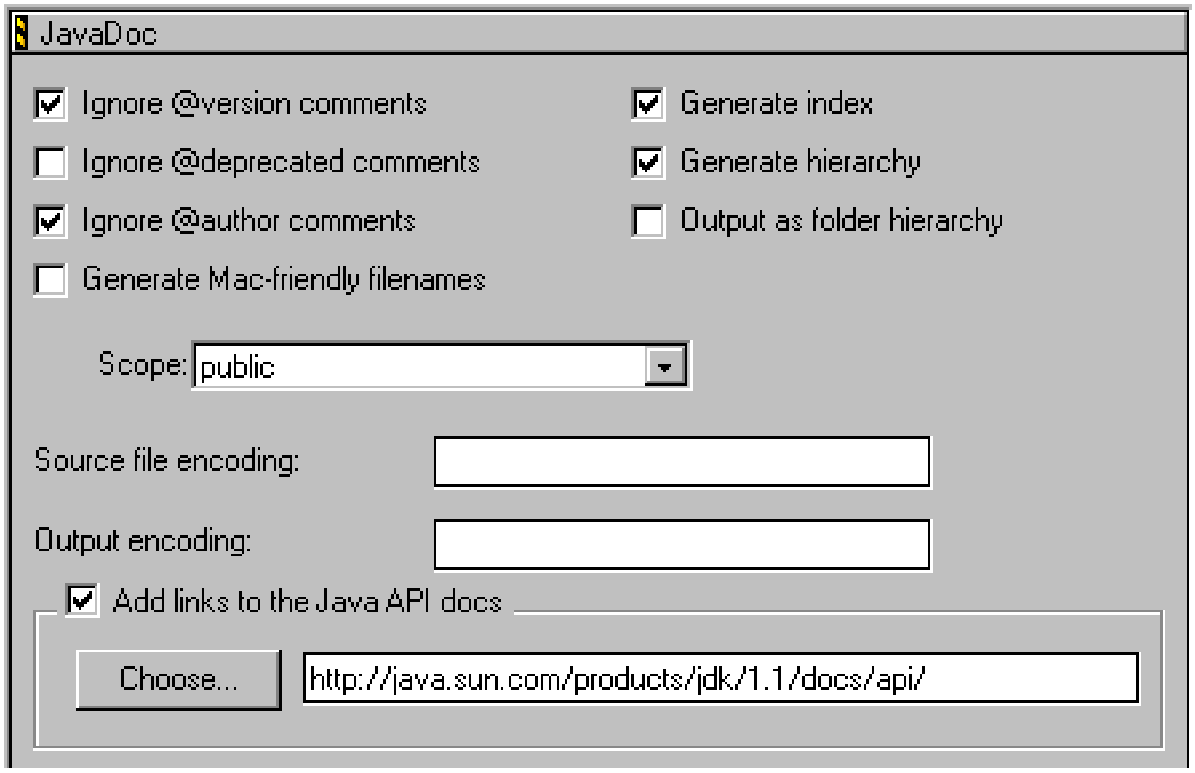
Figure 6.11 Output as Jar file



JavaDoc

The **JavaDoc** panel (Figure 6.12) controls how JavaDoc creates documentation for your Java source.

Figure 6.12 JavaDoc panel



Ignore options

Three ignore options: **Ignore @version comments**, **Ignore @deprecated comments**, and **Ignore @author comments** tell JavaDoc to omit comments with these "tags" from the final output when enabled.

Generate Mac-friendly filenames

Tells JavaDoc to generate filenames shortened to 31 characters to create cross-platform friendly HTML documents.

Generate index

Generate index tells JavaDoc to create index information in the final output. Sometimes, the index can be very large. Turn this option off if you do not want index information generated for your project.

Generate hierarchy

Generate hierarchy tells JavaDoc to generate class hierarchy information. Turn this option off if you do not want hierarchy information created.

Output as folder hierarchy

Instead of only outputting the HTML files for a project to a flat folder hierarchy with extremely long filenames, the JavaDoc Pre-Linker has the option of putting the files into a package-based folder hierarchy. The actual file names are only the class name.

Scope

The **Scope** popup menu controls which classes will be included in the documentation. Choices include:

- public
- protected and public
- protected, public and package
- all

NOTE: If you do not specifically declare a class to be public, private, or protected, it will automatically be declared as package.

Encoding options

The two encoding options: **Source file encoding** and **Output encoding** are for internationalization of your JavaDoc files. For example, the Java source files can be in Arabic and output in English.

Encoding numbers are used in each field. JavaDoc converts the number in the source field to Unicode, and then from Unicode to

the value specified in the output field. Leave these fields blank to use the default ISO Latin (859-1) encoding.

For more information on encoding, see:

`http://java.sun.com/products/jdk/1.1/intl/html/intlspec.doc7.html`

Add links to the Java API docs

This option adds links to the main Java API docs where appropriate in your code. The default link is on the Web. If you have a dial-up connection, you might prefer to have the API docs somewhere on your local machine. This way, you can specify the URL to be something like `file:///my_drive/jdk_docs/`. You can also specify a relative URL.

NOTE: If you move the docs around, the links may break.

7

Debugging Java Projects

The CodeWarrior debugger lets you debug Java code as easily as C/C++ or Pascal code. This chapter assumes you're already familiar with the CodeWarrior debugger. If you need more information about the debugger, see the *Debugger User Guide*. If you want a quick tutorial showing you how to use the debugger with Java, see "Programming Tutorial for Java" on page 19. If you want step-by-step information on running a Java project under the debugger, see "Debugging a Java Project" on page 63.

This chapter contains the following sections:

- Debugger Features and Limitations describes some of the features in the debugger that help you debug Java code and some limitations of the debugger.
- Special Debugger Features for Java goes into more detail about some special debugger features for Java.

NOTE: To debug Java source on Windows, yo must install Microsoft Internet Explorer or the Sun JDK. Both are included on your CodeWarrior CD.

Debugger Features and Limitations

The CodeWarrior Debugger contains many features that are specifically for debugging Java projects. It lets you:

- Debug Jar files, class files and applications.
- View a disassembly of the Java Virtual Machine instructions for your class, described in "Viewing the Java Virtual Machine Disassembly" on page 96.

- Debug multiple thread programs, as described in "Debugging Threads" on page 95.
- Break on Java exceptions, described in "Breaking on Java Exceptions" on page 94.

The debugger has a couple limitations that a future version will solve:

- Due to a limitation in the Sun 1.1.6 VM, Jar file debugging is not supported when using Suns VM .
- Multi-language debugging is not supported. When debugging a Java program with native C methods in it, the debugger treats it as a Java program and it will not be able to step into any native code.
- The debugger cannot debug compressed Jar or compressed zip files. This includes Java applications that are in compressed Jar or compressed zip format.

Special Debugger Features for Java

This section explains some of the special CodeWarrior Debugger features for Java. It contains the following:

- Breaking on Java Exceptions
- Opening Multiple Class Files in One Browser
- Choosing a Java Applet Viewer for Debugging
- Debugging Threads
- Viewing the Java Virtual Machine Disassembly
- Specifying Java Debugger Settings

Breaking on Java Exceptions

The CodeWarrior debugger can automatically break when your class throws an exception. Just choose one of these options from the **Break on Java exceptions** hierarchical menu in the Control menu.

If you choose	The debugger breaks on exceptions from…
All exceptions	Any class, including the Java API classes.
No exceptions	No classes.

Opening Multiple Class Files in One Browser

If you're debugging a project that creates multiple class files, you can view all the class files for a folder in one Browser window. Just turn on the **Open all class files in directory hierarchy** option in the Symbolics preference panel in the CodeWarrior Debugger. When you open a class file, the Browser window displays the classes for that file as well as all for the files in the same folder and its subfolders.

If that option is off, the Browser window displays the classes for one class file only.

Choosing a Java Applet Viewer for Debugging

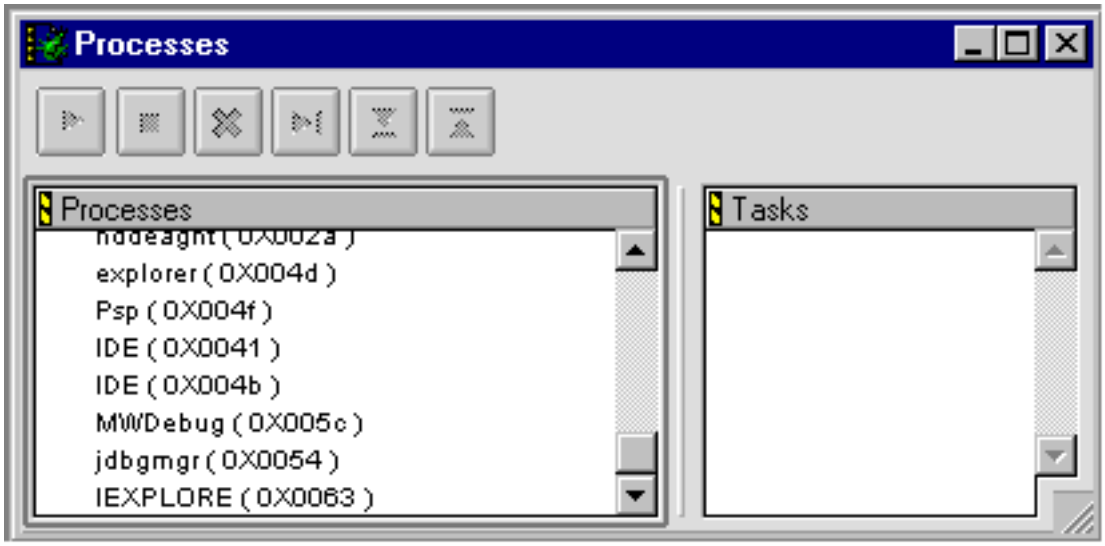
When debugging a Java applet, the applet runs within an applet viewer, such as a Java-aware HTML browser like the default, Internet Explorer. To learn how to specify the applet viewer, see "Applet" on page 77.

Debugging Threads

When you are debugging a Java program, the running process is either the specified applet viewer, or the Java interpreter, `jview.exe`.

You can view all the threads in a Java program in the Processes window. Just choose **Show Processes** from the Window menu and select the Metrowerks Java process in the left pane. The Java threads appear in the pane to the right.

Figure 7.1 The Processes window



To display a Program window for a thread, double-click it. To pause, continue, or kill a thread, select it and use the buttons in the Processes window.

NOTE: Many of the threads listed in the processes window are spawned by the applet viewer or interpreter running the Java program, or threads for the CodeWarrior Debugger. Pausing or killing one of them will have unforeseeable (and possibly quite unfortunate) consequences.

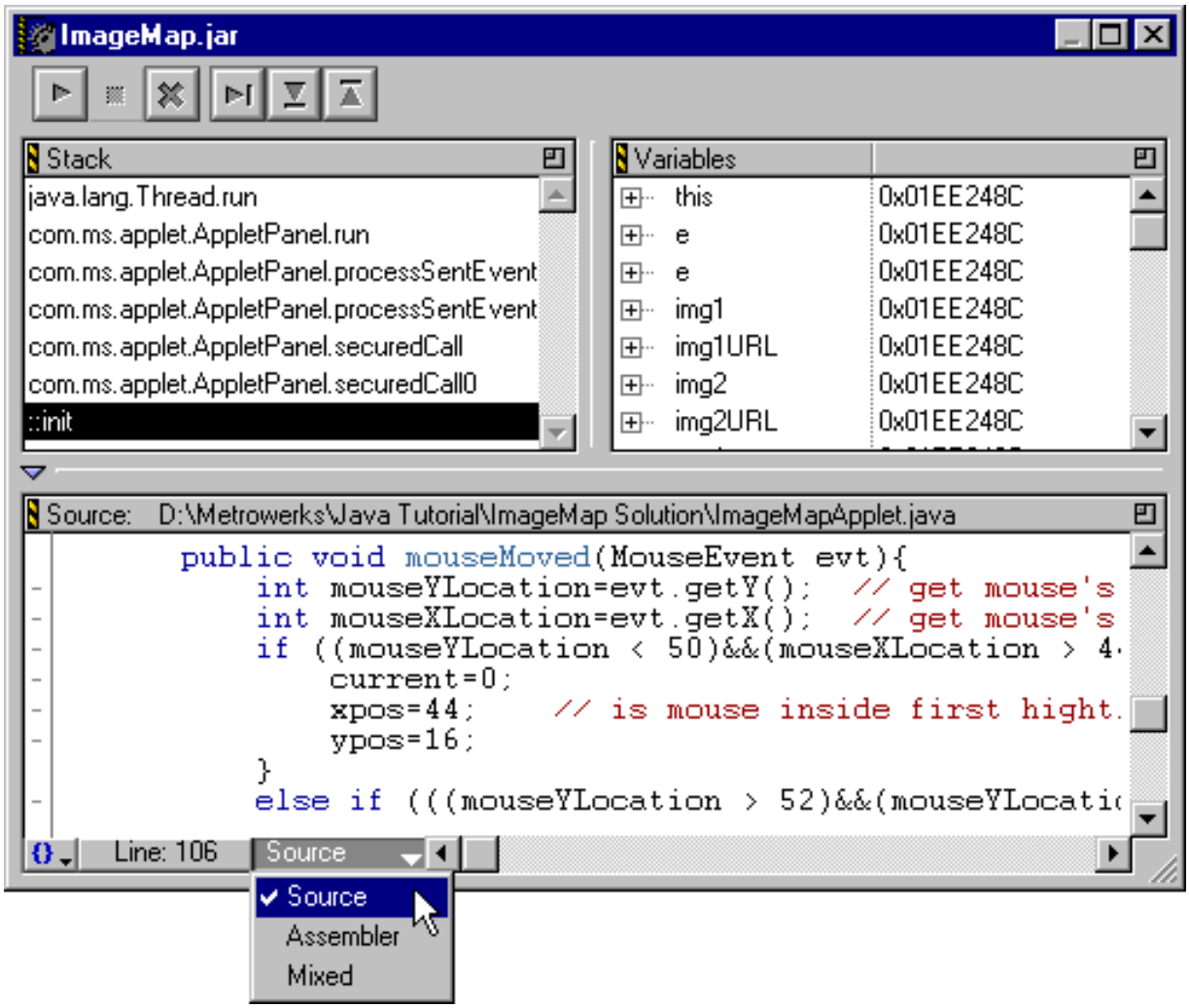
Viewing the Java Virtual Machine Disassembly

You can view a listing of the Java Virtual Machine instructions that implement the class. Just choose **Assembler** or **Mixed** from the Source pop-up menu at the bottom of any Browser or Program window (Figure 7.2).

The mixed view shows your Java source first, then the assembly language instructions that make up the Java code immediately after.

See also "Disassembling Classes" on page 134.

Figure 7.2 Viewing Java assembly



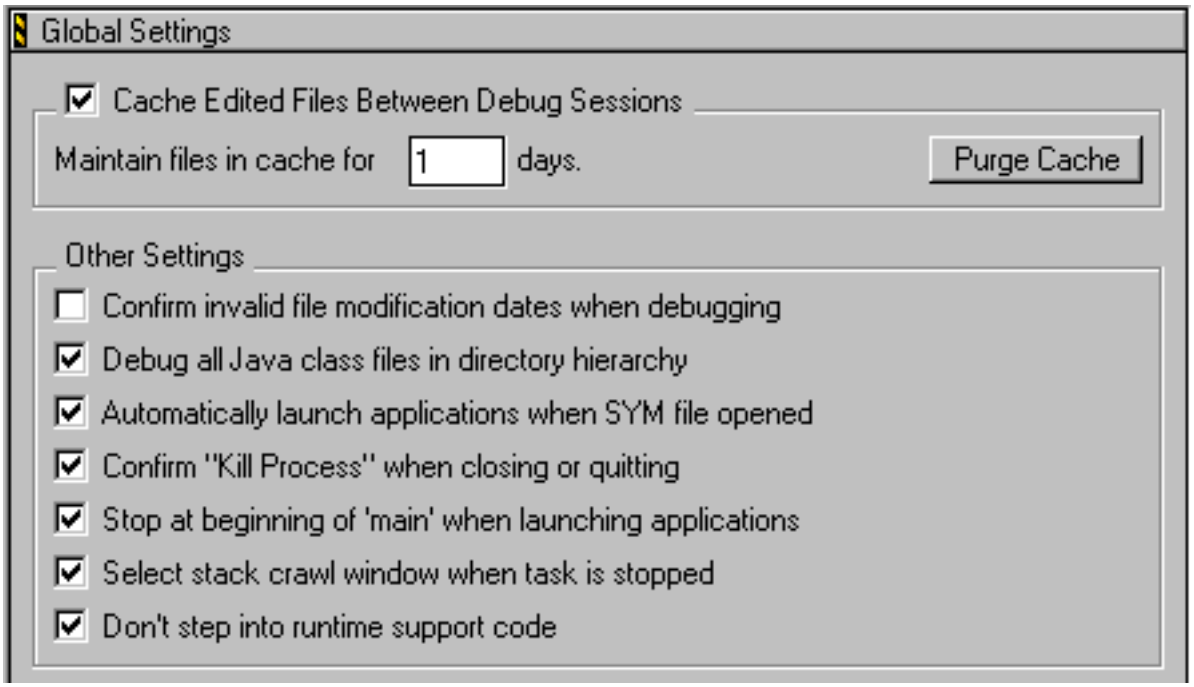
When viewing the assembly, you can still set breakpoints, step through code, and view variables. You can also view the assembly for a Java file in CodeWarrior by choosing **Disassemble** from the Project menu.

Specifying Java Debugger Settings

There is one setting in Global Preferences panel of the debugger group in the main IDE Preferences dialog that applies to Java: **Debug all Java class files in directory hierarchy**.

There are settings in the debugger that apply to Java: **Debug all Java class files in directory hierarchy** in the Global Preferences panel, and the settings in the Java Settings panel.

Figure 7.3 Symbolics preference panel

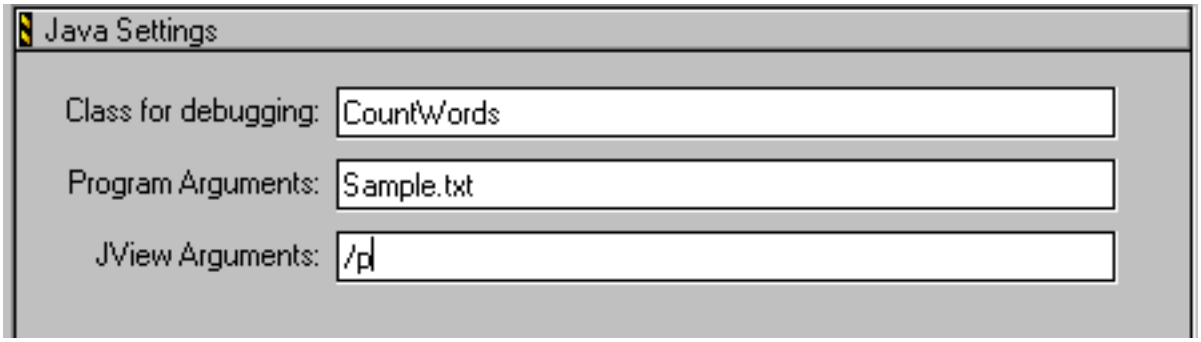

Global Settings
Cache Edited Files Between Debug Sessions
Maintain files in cache for
1
days.
Purge Cache
Other Settings
Confirm invalid file modification dates when debugging
Debug all Java class files in directory hierarchy
Automatically launch applications when SYM file opened
Confirm "Kill Process" when closing or quitting
Stop at beginning of 'main' when launching applications
Select stack crawl window when task is stopped
Don't step into runtime support code

The **Debug all class files in directory hierarchy** option opens all the class files in the directory and all contained directories, and merges them all together in the same Browser window.

Java Settings Panel

The Java Settings panel (Figure 7.4) controls how the debugger interacts with JView when debugging Java applets or applications.

Figure 7.4 The Debugger's Java Settings preferences panel



The items in this panel are:

Class for Debugging

This field specifies the particular class you wish to debug.

Program Arguments

This field specifies command-line arguments to be passed to your Java program when the debugger launches the program.

JView Arguments

This field specifies arguments to be passed to the jview interpreter by the debugger when it launches the Java application. Argument options are shown in Table 7.1.

Table 7.1 Jview argument options

Option	Description
`/?`	displays usage text
`/cp <classpath>`	set class path
`/cp:p <path>`	prepend path to class path
`/cp:a <path>`	append path to class path
`/n <namespace>`	namespace in which to run

Option	Description
`/p`	pause before terminating if an error occurs
`/v`	verify all classes
`/d:<name>=<value>`	define system property
`/a`	execute AppletViewer

8

Troubleshooting on Java

This section is designed to give you a quick reference point for common problems with their solutions when using CodeWarrior on Java. This should be the first place you look before contacting technical support.

Troubleshooting questions and answers cover the following main categories:

- Programming Problems
- Conversion Problems

Programming Problems

This section deals with problems encountered writing Java code using the CodeWarrior development environment.

There are no problems to report at this time. If you find you are having problems in this area, please send a bug report to Metrowerks support, `support@metrowerks.com`, and that information will be added to this manual in the next release.

Conversion Problems

This section deals mainly with problems encountered converting older CodeWarrior Java projects to the latest release.

There are no problems to report at this time. If you find you are having problems in this area, please send a bug report to Metrowerks support, `support@metrowerks.com`, and that information will be added to this manual in the next release.

Index

A

Add links to the Java API docs
    JavaDoc Project panel 91
All exceptions command 95
Applet
    defined 55
Applet project type
    Java Target panel 77
AppletFrame.java 57
applets
    choosing viewer 95
    debugging 94–97
Application
    defined 56
Application project type
    Java Target panel 77
applications
    debugging 94–97
Assembler view
    in MWDebug 96

B

BCI See bytecode interpreter
Binary Transfer
    FTP Post-Linker panel 86
bytecode
    compared to object code 13
    defined 12
bytecode interpreter defined 14

C

class files
    debugging 93–97
    disassembling 96
class folder 62
Class Folder option
    Java Output panel 87
Class folder option
    Name field 87
Class for Debugging
    Java Settings panel 99
`classes.zip` file 64
CodeWarrior
    integrated development environment (IDE) 15
    tutorials 10
Compress
    Java Output panel 88

D

Debug all class files in directory hierarchy (Debugger) 97, 98
Debugger User Guide 10, 17, 93
debugging Java code 93–97
Delete class files from output directory before linking
    Java Output panel 87
development tools
    CodeWarrior IDE (integrated development environment) 15
disassembling
    class files 96
documentation
    Debugger User Guide 10, 17, 93
    IDE User Guide 10, 15

E

encoding options
    JavaDoc Project panel 90
exceptions, breaking on 94

F

Folder to Upload
    FTP Post-Linker panel 86
FTP Post-Linker panel 84–86
    Binary Transfer 86
    Folder to Upload 86
    Generate Log 86
    Host Address 85
    Password 86
    Remote Directory 86
    User Name 86

G

Generate Coments in Headers
    Java Language panel 83
Generate Headers for All Methods
    Java Language panel 83
Generate hierarchy

JavaDoc Project panel 90
Generate index
JavaDoc Project panel 90
Generate Log
FTP Post-Linker panel 86
Generate Mac-friendly filenames
JavaDoc Project panel 89
Generate Manifest
Java Output panel 88

H

Host Address
FTP Post-Linker panel 85

I

IDE. *See* CodeWarrior, integrated development environment
IDE User Guide 10, 15
Ignore @author comments
JavaDoc Project panel 89
Ignore @deprecated comments
JavaDoc Project panel 89
Ignore @version comments
JavaDoc Project panel 89

J

Jar file 57, 62
defined 88
Jar File option
Java Output panel 88
Jar files 56
Java
bytecodes 12
overview 12
system requirements 11
tutorial 19–54
virtual machine 12
Java Language panel
Generate Comments in Headers 83
Generate Headers for All Methods 83
Java Output panel 86–??
Class folder option 87
compress 88
Delete class files from output directory before linking 87
Generate Manifest 88
Jar File option 88
Java platform 15
Java Settings panel 98
Class for Debugging 99
JView Arguments 99
Program Arguments 99
Java Target Panel ??–79
Java Target panel 76–??
Applet project type 77
Application project type 77
Library project type 79
Main Class field 78
Parameters field 78
JavaDoc 67–71
using 67–71
JavaDoc Project panel 89–91
Add links to the Java API docs 91
Encoding Options 90
Generate hierarchy 90
Generate Mac-friendly filenames 89
Ignore @author comments 89
Ignore @deprecated comments 89
Ignore @version comments 89
Output as folder hierarchy 90
Output encoding 90
Scope popup menu 90
Source file encoding 90
JavaDocProject panel
Generate index 90
JView Arguments
Java Settings panel 99

L

Library
defined 56
Library project type
Java Target panel 79
linker option 75

M

Mac OS platform 15
Main Class field

Java Target panel 78
Mixed view
in MWDebug 96
MWDebug
Assembler view 96
Mixed view 96

N

Name field
Class folder option 87
No exceptions command 95

O

object code
compared to bytecode 13
Open all class files in directory hierarchy 95
Output as folder hierarchy
JavaDoc Project panel 90
output directory option 76
Output encoding
JavaDoc Project panel 90

P

Parameters field
Java Target panel 78
Password
FTP Post-Linker panel 86
platforms
Java 15
Mac OS 15
Win32 15
Post-linker option 76
PowerPC processor 15
Pre-linker option 76
processors
PowerPC 15
68K 15
x86 15
Program Arguments
Java Settings panel 99
Project Stationery
described 56

R

Remote Directory
FTP Post-Linker panel 86

S

Scope popup menu
JavaDoc Project panel 90
Show Processes command (MWDebug) 95
68K processor 15
Source file encoding
JavaDoc Project panel 90
system requirements 11

T

target name option 75
Target Settings panel 74–76
Troubleshooting 101–102
tutorial 19–54
Tutorials, CodeWarrior 10

U

Unicode 90
User Name
FTP Post-Linker panel 86

V

virtual machine
and platform-independent code 12
defined 12

W

Win32 platform 15

X

x86 processor 15

Z

zip files 56

CodeWarrior

Targeting the Java VM

From Windows

Credits

writing lead: David Blache

other writers: L. Frank Turovich, Marc Paquette, Chris Magnuson, Derek Saldana, Jim Trudeau

engineering: Greg Bolsinga, Kevin Buettner, John Cortell, Bernie Estavillo, Michael Farrar, Tim Freehill, Michael Stricklin

frontline warriors: Greg Bolsinga, Barry Cameron, Pedro Larios, David Mark, Michael Stricklin, Ferry Sutanto, and CodeWarrior users everywhere

Guide to CodeWarrior Documentation

CodeWarrior documentation is modular, like the underlying tools. There are manuals for the core tools, languages, libraries, and targets. The exact documentation provided with any CodeWarrior product is tailored to the tools included with the product. Your product will not have every manual listed here. However, you will probably have additional manuals (not listed here) for utilities or other software specific to your product.

Core Documentation	
IDE User Guide	How to use the CodeWarrior IDE
Debugger User Guide	How to use the CodeWarrior debugger
CodeWarrior Core Tutorials	Step-by-step introduction to IDE components
Language/Compiler Documentation	
C Compilers Reference	Information on the C/C++ front-end compiler
Pascal Compilers Reference	Information on the Pascal front-end compiler
Error Reference	Comprehensive list of compiler/linker error messages, with many fixes
Pascal Language Reference	The Metrowerks implementation of ANS Pascal
Assembler Guide	Stand-alone assembler syntax
Command-Line Tools Reference	Command-line options for Mac OS and Be compilers
Plugin API Manual	The CodeWarrior plugin compiler/linker API
Library Documentation	
MSL C Reference	Function reference for the Metrowerks ANSI standard C library
MSL C++ Reference	Function reference for the Metrowerks ANSI standard C++ library
Pascal Library Reference	Function reference for the Metrowerks ANS Pascal library
MFC Reference	Reference for the Microsoft Foundation Classes for Win32
Win32 SDK Reference	Microsoft's Reference for the Win32 API
The PowerPlant Book	Introductory guide to the Metrowerks application framework for Mac OS
PowerPlant Advanced Topics	Advanced topics in PowerPlant programming for Mac OS
Targeting Manuals	
Targeting Java	How to use CodeWarrior to program for Java
Targeting Mac OS	How to use CodeWarrior to program for Mac OS
Targeting MIPS	How to use CodeWarrior to program for MIPS embedded processors
Targeting NEC V810/830	How to use CodeWarrior to program for NEC V810/830 processors
Targeting Net Yaroze	How to use CodeWarrior to program for Net Yaroze game console
Targeting Nucleus	How to use CodeWarrior to program for the Nucleus RTOS
Targeting OS-9	How to use CodeWarrior to program for the OS-9 RTOS
Targeting Palm OS	How to use CodeWarrior to program for PalmPilot
Targeting PlayStation OS	How to use CodeWarrior to program for the PlayStation game console
Targeting PowerPC Embedded Systems	How to use CodeWarrior to program for PPC embedded processors
Targeting VxWorks	How to use CodeWarrior to program for the VxWorks RTOS
Targeting Win32	How to use CodeWarrior to program for Windows